MILENA ALMEIDA LEITE BRANDÃO

EVOLUÇÃO DIFERENCIAL MELHORADA IMPLEMENTADA EM PROCESSAMENTO PARALELO

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
2014

MILENA ALMEIDA LEITE BRANDÃO

EVOLUÇÃO DIFERENCIAL MELHORADA IMPLEMENTADA EM PROCESSAMENTO PARALELO

Tese apresentada ao Programa de Pós-graduação em Engenharia Mecânica da Universidade Federal de Uberlândia, como parte dos requisitos para a obtenção do título de **DOUTOR EM ENGENHARIA MECÂNICA**.

Área de concentração: Mecânica dos sólidos e vibração.

Orientadora: Profa. Dra. Sezimária de Fátima Pereira Saramago

Uberlândia - MG

2014

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

B817e 2014

Brandão, Milena Almeida Leite, 1985-
Evolução diferencial melhorada implementada em processamento paralelo / Milena Almeida Leite Brandão. - 2014.
158 f. : il.

Orientadora: Sezimária de Fátima Saramago.

Tese (doutorado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Mecânica.
Inclui bibliografia.

1. Engenharia mecânica - Teses. 2. Otimização combinatória - Teses. 3. Automação industrial - Teses. 4. Sistemas lineares - Teses. I. Saramago, Sezimária de Fátima. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Engenharia Mecânica. III. Título.

CDU: 62

Dedico este trabalho ao meu marido Jordaneo que sempre me apoiou. Te amo muitoooo!

AGRADECIMENTOS

Ao meu Deus Jeová por me abençoar e me guiar;

À Universidade Federal de Uberlândia e à Faculdade de Engenharia Mecânica;

À minha orientadora Prof. Dra. Sezimária de Fátima Pereira Saramago (Sezi) pela ajuda na escolha do tema desse trabalho, confiança em me orientar, paciência, amizade, respeito e por me passar suas experiências e aprendizados sem reservas. Sentirei saudades Sezi;

Ao Prof. Dr. José Laércio Doricio que trabalhou arduamente comigo na implementação dos códigos computacionais desempenhando o papel de um co-orientor. Obrigada Laércio pela paciência e amizade;

À CAPES pelo apoio financeiro através da bolsa de estudos;

Ao Laboratório de Computação Científica Aplicada e Tecnologia de Informação - LCCATI da FACIP/UFU por possibilitar simulações num cluster robusto;

Ao Laboratório de Mecânica dos Fluidos e Análise Numérica da FACIP - LMFAN/FACIP;

Aos meus pais Neide e Anderson pelo grande apoio emocional. Sempre me fizeram sentir mais inteligente do que realmente sou;

Ao meu marido Jordaneo pela extrema paciência. Desde que me conheceu me acompanha nesta rotina frenética de estudos;

À minha irmã Núbia, que por seguir os meus passos, me motivou a sempre continuar;

Aos professores Francisco José da Cunha Pires Soeiro, Paulo Roberto Bergamaschi e Valder Steffen Júnior por terem aceito o convite para participarem da banca examinadora;

Aos amigos do doutorado, principalmente à Ana Paula e ao Thiago De Paula Sales;

Aos amigos de trabalho do Curso de Graduação em Matemática da FACIP/UFU, pela compreensão e companherismo;

Para finalizar, estendo meus agradecimentos aos demais colegas de trabalho, professores e técnicos da FEMEC/UFU, amigos e familiares.

BRANDÃO M. A. L., **Evolução Diferencial Melhorada Implementada em Processamento Paralelo.** 2014. 158f. Tese de Doutorado, Universidade Federal de Uberlândia, Uberlândia.

RESUMO

O objetivo deste trabalho é apresentar um aprimoramento do método de otimização heurístico Evolução Diferencial propondo modificações no seu algoritmo básico através da utilização do conceito de Evolução com Conjuntos Embaralhados. Desenvolveu-se o método denominado Evolução Diferencial Melhorada (EDM), sendo este algoritmo implementado em computação paralela (EDMP), tornando-o apto para solucionar problemas de otimização complexos. O algoritmo desenvolvido é adaptado para trabalhar com problemas de otimização multiobjetivos e na presença de restrições. Algumas funções testes são resolvidas usando o método EDMP a fim de validar o algoritmo. A metodologia EDMP é utilizada para obter o projeto ótimo de um robô manipulador com três juntas rotacionais (3R) levando-se em conta as características de sua topologia. Com esta finalidade, um problema de otimização multiobjetivo é formulado para a obtenção dos parâmetros geométricos ótimos do robô, considerando a maximização do volume do espaço de trabalho, de sua rigidez e a otimização de sua destreza. Finalmente, o algoritmo EDMP é aplicado na solução de grandes sistemas lineares, reescritos como um problema de minimização de resíduos. Todos os resultados obtidos com o algoritmo desenvolvido são comparados com as soluções calculadas por meio de outras metodologias, a fim de comprovar sua eficiência e o relevante ganho em termos de tempo computacional.

Palavras Chave: Otimização, Evolução Diferencial Melhorada, Computação Paralela, Robô Manipulador, Grandes Sistemas Lineares.

BRANDÃO M. A. L., **Improved Differential Evolution Implemented in Parallel Processing.** 2014. 158f. Doctorate Thesis, Universidade Federal de Uberlândia, Uberlândia.

ABSTRACT

The aim of this work is to present an improvement of the heuristic optimization method of Differential Evolution proposing modifications to its basic algorithm by using the concept of evolution with Shuffled Sets. The method called Improved Differential Evolution (IDE) was developed, and implemented for parallel computing (IDEP), making it suitable for solving complex optimization problems. The algorithm developed is adapted to work with multiple objective optimization problems and in the presence of constraints. Some test functions are solved by using the IDEP method in order to validate the algorithm. The IDEP methodology is utilized to obtain the optimal design of a robot manipulator with three rotational joints (3R) taking into account the characteristics of its topology. For this purpose, a multiple objective optimization problem is formulated to obtain optimum robot geometrical parameters, considering the maximization of the volume of the workspace, the rigidity and the optimization of its dexterity. Finally, the IDEP algorithm is applied to solve large linear systems, rewritten as a residues minimization problem. All the results obtained with the developed algorithm are compared with the solutions calculated through other methodologies in order to prove its efficiency and the relevant gain in terms of computational time.

Keywords: Optimization, Improved Differential Evolution, Parallel Computing, Robot Manipulator, Large Linear Systems.

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

AG	-	Algoritmo Genético
CR	-	Probabilidade de cruzamento em Evolução Diferencial
D	-	Conjunto de índices das restrições de desigualdade
D_f	-	Domínio da função f
E_p	-	Eficiência da execução de um programa em paralelo com p processadores
ECC	-	Evolução com Conjuntos Competitivos
ECE	-	Evolução com Conjuntos Embaralhados
ED	-	Evolução Diferencial
EDM	-	Evolução Diferencial Melhorada
$EDMP$	-	Evolução Diferencial Melhorada Implementada em Paralelo
f	-	Função objetivo do problema de otimização
F	-	Taxa de pertubação em Evolução Diferencial
f^0	-	Vetor objetivo ideal do problema de otimização
$GMRES$	-	Método Iterativo Resíduo Mínimo Generalizado
$GPUs$	-	Graphics Processing Unit (Unidade de Processamento Gráfico)
HPC	-	Computação de alto desempenho (High Performance Computing)
I	-	Conjunto de índices das restrições de igualdade
k	-	k-ésima iteração do algoritmo
max	-	Maximizar
min	-	Minimizar
MPI	-	Message Passing Interface
N_p	-	Quantidade de indivíduos na populção inicial em Evolução Diferencial
PC	-	Computadores pessoais (Personal Computer)
p_c	-	Probabilidade de cruzamento em Algoritmo Genético
p_m	-	Probabilidade de mutação em Algoritmo Genético
$rand$	-	Número aleatório uniforme entre 0 e 1
S_p	-	Speedup da execução de um programa em paralelo com p processadores
x^*	-	Solução ótima do problema de otimização
x^0	-	Solução ideal do problema de otimização
σ	-	Desvio padrão
ω_i	-	Coeficiente de ponderação
$[-]^T$	-	Vetor Transposto

SUMÁRIO

CAPÍTULO I

INTRODUÇÃO

Os Algoritmos Evolucionários (AEs) são, em geral, métodos de otimização de busca estocástica que possuem como base da sua formulação a teoria da evolução. Nas últimas décadas o grande interesse dos pesquisadores por estes algoritmos tem impulsionado o desenvolvimento nos estudos o que tem levado a uma significativa melhora na eficiência e aplicabilidade destes métodos para resolver problemas complexos de otimização nas diferentes áreas do conhecimento.

Um algoritmo pertencente à classe dos AEs que tem se destacado é o método de otimização conhecido como Evolução Diferencial (ED). O algoritmo da ED tem se apresentado robusto e eficiente ao ser testado com sucesso em vários campos da ciência, tais como: solução de problemas multiobjetivo de controle ótimo com a flutuação de índice aplicado ao processo de fermentação (LOBATO et al (2007)), projeto de filtro-IIR parâmetro 18 (STORN (1995)), otimização multiobjetivo de projetos de sistemas de engenharia (LOBATO e STEFFEN (2007)), estimativa da difusividade térmica de secagem de frutas (MARIANI; LIMA; COELHO (2008)), no dimensionamento ótimo de tubos de distribuição de redes de água (SURIBABU (2010)), no projeto de um trocador de energia térmica para estimar a área mínima de transferência de calor (BABU e MUNAWAR (2007)), na minimização de custos do projeto ideal de vigas de concreto protendido (QUARANTA; FIORE; MARANO (2014)). No artigo de MENDES et al (2006),

devido a quantidade muito elevada de possíveis soluções ótimas, Evolução Diferencial é utilizado para resolver um problema de projeto de redes de rádio (Radio Network Design), que consiste em determinar os locais ideais para a base dos transmissores de estação, a fim de obter uma área de cobertura máxima com um número mínimo de transmissores. Outras aplicações podem ser vistas em DAS e SUGANTHAN (2011), STORN; PRICE; LAMPINEM (2005), STORN e PRICE (1997).

Por ser uma técnica relativamente nova, ED surgiu em meados da década de 90, muitos pesquisadores têm proposto modificações no algoritmo original da ED com intuito de melhorar sua convergência. Por meio de testes com problemas clássicos de otimização percebeu-se que às vezes os resultados obtidos com a ED não são tão bons quanto o esperado. Além disso, em muitos casos o algoritmo encerra a busca pela solução ótima prematuramente.

No trabalho de RAUPP; FAMPA; MELO (2010), é apresentada uma nova metaheurística baseada em Evolução Diferencial para o problema geral de programação não linear denominada Evolução Diferencial Aperfeiçoada (EDA). Contando com inovações no uso de operadores de cruzamento, mutação e seleção a EDA introduz um novo mecanismo para aumentar a diversidade no processo de exploração do espaço de busca, juntamente com um aperfeiçoamento no critério de seleção. Além disso, EDA introduz um critério de parada adicional que permite a detecção de convergência da população, o qual evita operações computacionais desnecessárias e, consequentemente, aumenta sua eficiência.

No artigo de SWAGATAM DAS; KONAR; CHAKRABORTY (2005) são propostas duas modificações no esquema básico da Evolução Diferencial: primeiro é introduzido o conceito de fator de escala tempo-variável pelo qual o vetor de diferença será multiplicado e segundo, o fator de escala é variado de um modo aleatório. O objetivo destas modificações é tentar evitar ou desacelerar a convergência prematura nos estágios iniciais da busca e facilitar a convergência para a solução global ótima durante as fases posteriores da pesquisa. Os resultados obtidos são comparados com outros métodos heurísticos usando sete funções testes mostraram que o novo esquema melhora, significamente, o desempenho do método. As seguintes métricas de desempenho foram

utilizadas: (a) a qualidade da solução, (b) a velocidade de convergência, (c) a frequência em encontrar o ponto ótimo, e (d) a escalabilidade. Com as variações propostas por SWAGATAM DAS; KONAR; CHAKRABORTY (2005), ED satisfez ou superou os resultados dos métodos concorrentes analisados na maioria dos casos.

LI et al (2011) descrevem a seguinte modificação no algoritmo da Evolução Diferencial: o vetor de mutação é gerado à partir de um vetor subótimo e de outros dois indivíduos da população. O novo algoritmo é chamado de Improved Differential Evolution (IDE). O objetivo desta alteração é acelerar a velocidade de convergência do método sem que ocorra uma convergência prematura. O IDE não introduz parâmetros extra de controle ao algoritmo original e o custo computacional da busca do individuo adicional subótimo é negligenciável. Os resultados da simulação com funções teste mostraram que o desempenho da IDE é superior ao da ED.

A fim de aumentar a precisão da solução ótima, no artigo de CHONG et al (2012), o novo algoritmo desenvolvido, IDE, é um algoritmo híbrido aplicando ED e Filtro de Kalman conjuntamente. Os resultados obtidos com IDE mostraram ser superiores aos obtidos com Algoritmo Genético e ED. Além disso, IDE gastou menos tempo de computação e alcançou maior precisão para resultados simulados em comparação ao AG e ED.

No trabalho de COELHO; SOUZA; MARIANI (2009), é apresentada uma nova abordagem para ED inspirado em uma medida de diversidade da população para o ajuste das taxas de cruzamento, seleção e mutação.

Estes são apenas alguns exemplos dos esforços da comunidade acadêmica em melhorar a convergência global da ED. Note que muitas modificações propostas envolvem o mesmo princípio, o de alterar algum aspecto das operações de mutação, cruzamento e seleção da ED.

Embora os AEs sejam algoritmos que possam resolver uma grande variedade de problemas de otimização, para solucionar problemas complexos de otimização o algoritmo pode demandar um elevado esforço computacional, exigindo muito tempo de processamento. Recentemente, com o avanço e disponibilidade das tecnologias computacionais tem-se pensando na implementação de algoritmos de otimização em paralelo

com o intuito de diminuir este tempo de processamento.

No trabalho de KWEDLO e BANDURSKI (2006) é sugerido um novo esquema de paralelização para o cálculo do valor da função objetivo baseado na decomposição de dados, nesta abordagem tanto o conjunto de aprendizagem (learning set) e a população do algoritmo evolutivo são distribuídos entre os processadores. Os processadores formam um gasoduto usando a topologia de anel. Num único passo cada processador calcula o valor da função objetivo da sua subpopulação enquanto envia a subpopulação anterior ao processador sucessor e recebe a próxima subpopulação do processador antecessor. De acordo com os autores, desta forma é possível sobrepor a comunicação e a computação. Os primeiros resultados experimentais mostram que para grandes conjuntos de dados o algoritmo obtem um excelente ganho de velocidade de processamento.

No artigo de TASOULIS et al (2004) ED é paralelizado em um ambiente paralelo virtual a fim de melhorar a velocidade e o desempenho do método. Segundo os autores os resultados experimentais indicam que a troca de informações entre as subpopulações atribuídas a diferentes processadores tem um impacto significante no desempenho do algoritmo.

Segundo BUJOK (2011), o principal problema da otimização está na complexidade e no tempo necessário para encontrar o ótimo global. Para ele, uma forma de resolver isto é por meio do uso de modelos paralelos do algoritmo da ED. Ainda, segundo o autor, existem diversas abordagens de paralelismo que podem conduzir à busca mais rápida pelo mínimo global. Os modelos paralelos mais utilizados são modelo mestre-escravo, modelo de ilha ou migração, modelo de difusão (celular) e a combinação destes resultam em modelos híbridos. No seu artigo é usado o modelo de migração para paralelizar ED da seguinte forma: a população é dividida em várias subpopulações (ilhas) e os indivíduos migram entre elas.

No artigo de CHI (2010) é demonstrado como aplicar o MapReduce e o cógico aberto Hadoop para uma paralelização fácil e rápida do algoritmo ED. De acordo com DEAN e GHEMAWAT (2014), MapReduce é um modelo de programação e implementação projetado para processar e gerar grandes volumes de dados em paralelo, dividindo o trabalho em conjunto de tarefas independentes. O Hadoop, de acordo com o projeto

APACHE^{TM} HADOOP® (2014), é uma plataforma de software de código aberto para computação distribuída voltada para clusters e processamento de grandes quantidades de dados. No artigo de CHI (2010) é sugerido aplicar o MapReduce somente na etapa de avaliação da função objetivo, que geralmente é o momento que consome a maior parte do tempo de execução do programa. Assim, não é feito uma paralelização de todo o processo de evolução.

Também há pesquisas onde o algoritmo Evolução Diferencial em paralelo é desenvolvido para cluster de computadores em ambiente Windows. Em NTIPTENI; VALAKOS; NIKOLOS (2006) a paralelização é realizada aplicando uma abordagem assíncrona, utilizando arquitetura mestre-escravo. Um programa executável separado é usado para evoluir cada membro da população. A população atual é armazenada em uma pasta acessível por todos os programas executáveis, cada membro da população atual, juntamente com a sua aptidão, é armazenado em um arquivo de texto separado contido nesta pasta comum. Cada programa escravo usa as informações armazenadas nesta pasta comum para evoluir o membro correspondente da população e para atualizar as informações armazenadas no arquivo de texto correspondente, independentemente de outros programas executáveis. Nesta metodologia, pode ser atribuído a cada computador mais de um programa executável.

No trabalho de QIANG (2013) um novo otimizador multiobjetivo em paralelo baseado no algoritmo da ED é proposto para otimizar a dinâmica de feixe photoinjetor (photoinjetor é um dispositivo que gera feixe de elétrons). O método é composto por dois níveis de paralelização. Primeiro a população é distribuída entre os processadores em paralelo; em seguida, cada avaliação da função objetivo corresponde a um acelerador da simulação.

Os trabalhos aqui mencionados confirmam os esforços da comunidade acadêmica em aprimorar as características do algoritmo ED. Tais esforços se concentram não apenas em melhorar a qualidade da convergência do método mas também na sua velocidade de convergência.

Visto que muitos problemas de otimização das engenharias são complexos, isto é, possuem funções descontínuas e/ou domínios não convexos, os métodos clássicos de

otimização que utilizam derivadas não tem apresentado resultados satisfatórios. Por isso há a necessidade de um algoritmo de otimização eficiente e robusto.

O algoritmo da ED possui excelentes características tais como: capacidade de escapar de ótimos locais, flexibilidade em lidar com problemas para os quais as funções podem ser não contínuas, não diferenciáveis e multimodais. Entretanto, de acordo com TOMASSINI (1999), algoritmos evolucionários podem se tornar muito lentos quando imposto a problemas de maior complexidade.

Desta forma, esta tese tem como objetivo principal apresentar um aprimoramento do método de otimização evolutivo Evolução Diferencial, propondo modificações no algoritmo básico através da utilização de conjuntos embaralhados (shuffled complex) e tornando-o capaz de trabalhar com processamento paralelo.

Tem-se como objetivos específicos estudar o desempenho computacional do novo algoritmo implementado, chamado Evolução Diferencial Melhorada implementado em Paralelo (EDMP), aplicado à solução dos seguintes problemas mecânicos: na otimização de sistemas robóticos e na solução de grandes sistemas lineares por meio da minimização do vetor resíduo $r = Ax - b$.

Em sistemas robóticos uma preocupação pertinente é de criar sistemas com propriedades mais eficientes e adaptáveis às necessidades do ambiente no qual o robô atuará. Neste trabalho será considerado os robôs manipuladores que são dispositivos programáveis projetados para executar uma grande variedade de tarefas de forma repetitiva. De acordo com SANTOS; SARAMAGO; STEFFEN (2005), em ambientes industriais, a diminuição de exigências dinâmicas associadas à realização de uma mesma tarefa de um robô manipulador, pode resultar em aumento de produtividade e diminuição dos custos associados à operação e manutenção do robô. Ainda, SANTOS; SARAMAGO; STEFFEN (2005) comentam que a diminuição destas exigências pode viabilizar a realização de tarefas que exijam a capacidade máxima do sistema, aumentando assim a versatilidade dos robôs para que possam se adequar a diversas situações. Além disso, segundo SANTOS; SARAMAGO; STEFFEN (2005), dentre os objetivos mais importantes da automação industrial, está a redução do custo de produção e o aumento da produtividade. Portanto, para viabilizar o uso de sistemas robóticos, é de grande importância

que se considere o planejamento e otimização da trajetória a ser executada.

A trajetória de um robô manipulador serial é determinada pelo movimento do efetuador a partir de um ponto inicial a um final sem preocupação com os pontos intermediários durante o percurso. Neste trabalho, o projeto ótimo de um robô manipulador corresponderá àquele que otimiza algumas características desejáveis para melhorar a performance do robô ao executar suas tarefas. A fim de se obter tal projeto ótimo será utilizado um método heurístico de otimização implementado em paralelo.

Por outro lado, sistemas lineares têm sido utilizados para modelar diversos fenômenos físicos e da natureza. Existem vários métodos de solução de sistemas lineares e a escolha de um deles depende diretamente das características do sistema. No entanto, muitos destes métodos tratam apenas de sistemas lineares onde a matriz dos coeficientes é simétrica e positiva definida. Neste trabalho aplicar-se-á técnicas heurísticas de otimização para encontrar a solução de sistemas lineares sem se preocupar com as características da matriz de coeficientes. Além disso, não será necessária a utilização de pré-condicionadores para adequar o sistema aos métodos tradicionais. Desta forma, este trabalho poderá contribuir fornecendo uma alternativa à escolha de métodos para solucionar sistemas lineares de grande porte.

A tese está dividida do seguinte modo:

Capítulo 2: Formulação matemática do problema de otimização uni e multiobjetivo e classificação dos métodos de otimização.

Capítulo 3: É apresentado uma breve descrição do surgimento das estratégias evolutivas e, em seguida, são consideradas as definições e principais características dos métodos de otimização: Algoritmo Genético, Evolução Diferencial, Evolução com Conjuntos Embaralhados, Evolução com Conjuntos Competitivos, Evolução Diferencial Melhorada e Evolução Diferencial Melhorada utilizando Processamento Paralelo.

Capítulo 4: Trata-se da computação paralela como uma forma de melhorar o desempenho de um programa que pode ser executado em um super computador ou em

um cluster. São apresentados as principais características, construção e funcionamento de um cluster Beowulf. São consideradas também algumas métricas de desempenho que serão utilizadas neste estudo para avaliar um processamento paralelo.

Capítulo 5: Este capítulo é voltado para as simulações numéricas realizadas neste trabalho. Inicialmente, a fim de comparar alguns métodos de otimização é obtido o projeto ótimo de um recipiente de pressão e de uma viga engastada com os métodos Algoritmo Genético, Evolução Diferencial e Evolução com Conjuntos Embaralhados. Em seguida, algumas funções testes são utilizadas para verificar a capacidade de otimização do código computacional Evolução Diferencial Melhorada implementado em paralelo. Além disso, um problema prático da engenharia mecânica que consiste em obter o projeto ideal de um robô manipulador levando em conta as características do seu espaço de trabalho é modelado e resolvido por meio do algoritmo Evolução Diferencial Melhorada implementada em paralelo. Finalmente, o algoritmo da Evolução Diferencial Melhorada implementada em paralelo é aplicado à solução de dois problemas de identificação de forças dinâmicas por meio de grandes sistemas lineares.

Capítulo 6: São apresentadas algumas conclusões sobre este estudo e propostos alguns temas para trabalhos futuros.

CAPÍTULO II

FUNDAMENTOS BÁSICOS - PROBLEMAS DE OTIMIZAÇÃO

A otimização matemática é um ramo da matemática e da ciência computacional que desenvolve estudos e ferramentas analíticas e numéricas para modelar e resolver problemas de otimização. Assim, a otimização lida com problemas de encontrar mínimos, máximos ou zeros de funções. Tais problemas surgem, por exemplo, nas áreas da matemática, das ciências físicas, químicas e biológicas, engenharia, arquitetura, economia e gestão.

Mas não é só nas ciências que se encontram problemas de otimização. Otimização é uma realidade que se aplica todos os dias, em quase todos os assuntos. Até mesmo os animais de uma forma instintiva aplicam a otimização no seu dia-a-dia. Um exemplo clássico disto são as abelhas. Esses insetos usam cera para construir os alvéolos das colméias, os quais são usados como depósitos para o mel que fabricam. Os alvéolos que compõem o favo de mel das abelhas européias possuem um formato poliédrico. Mas por que não circular, já que o círculo é a forma geométrica que tem maior volume com menor área? Porque as abelhas constroem os alvéolos procurando a forma mais econômica, ou seja, que apresente maior volume para a menor porção de material gasto. Como os alvéolos não poderiam ser, por exemplo, cilíndricos, pois a falta de paredes comuns entre eles deixaria uma grande quantidade de espaços inaproveitáveis, a solução encontrada pelas abelhas é que os alvéolos devem ter a forma de um

prisma, pois assim, as paredes de um alvéolo servem também aos alvéolos vizinhos, desta forma conseguem guardar um maior volume de mel com a mínima quantidade de material gasto. Este é apenas um dos vários exemplos encontrados na natureza.

A otimização é uma ferramenta importante para a tomada de decisão durante a análise e o projeto de sistemas físicos. Para utilizar esta ferramenta torna-se necessário identificar qual o objetivo, ou seja, a quantidade que irá medir a performance do sistema (ex: energia, custo, potência, tempo, temperatura, etc.). O objetivo depende das características do sistema, que são as *variáveis* ou *parâmetros*, denominados variáveis de projeto ou variáveis de decisão. Assim, a otimização visa encontrar o valor das variáveis para otimizar o objetivo. Normalmente, estas variáveis devem obedecer a restrições, que são limites impostos ao sistema estudado.

O processo de identificação dos objetivos, variáveis e restrições é denominado *modelagem* do problema. Escrever um modelo adequado é muitas vezes o passo mais importante do processo de otimização. Se o modelo é muito simples, ele pode não representar o problema real. Por outro lado, se o modelo é muito complexo, ele pode trazer muitas dificuldades para a obtenção da solução.

Após a formulação do modelo, um algoritmo de otimização é utilizado para obter a solução do problema, usualmente com a ajuda de códigos computacionais. Não existe um algoritmo universal, mas um conjunto de métodos, entre os quais alguns são mais apropriados para determinadas aplicações específicas. A escolha do método de otimização depende do usuário e pode determinar a eficácia ou falha para a solução do problema.

Após a aplicação do algoritmo o projetista deve ser capaz de analisar a solução encontrada procurando identificar se o processo de otimização obteve sucesso ou se falhou. Em uma análise teórica, existem expressões matemáticas elegantes conhecidas como condições necessárias para existência do ponto ótimo (optimality condition) para verificar se a solução atual representa a solução ótima local do problema, mas nos problemas reais elas são difíceis de serem verificadas.

O modelo pode ser aperfeiçoado usando “análise de sensibilidade” que permite avaliar quais variáveis de projeto são mais relevantes no modelo que representa o

problema.

Um problema de otimização é frequentemente chamado de progamação matemática. Os algoritmos de otimização são *iterativos*. Iniciam com uma estimativa inicial x^0 e geram uma sequência de aproximações até encontrar o ponto ótimo. As estratégias utilizadas para se mover de uma iteração para outra é que distingue os diversos algoritmos. Muitas estratégias (métodos clássicos) usam o valor da função objetivo e suas derivadas. Algumas estratégias armazenam informações de iterações anteriores e outras consideram a informação apenas da iteração atual. De acordo com NOCEDAL e WRIGHT (2000), bons algoritmos de otimização devem ter as seguintes propriedades:

Robustez: devem ter uma boa performance para uma grande variedade de problemas e obter resultados razoáveis para quaisquer pontos de partida.

Eficiência: não devem exigir tempo computacional excessivo ou grande capacidade de armazenamento de dados.

Precisão: devem ser capazes de identificar a solução com precisão, sendo pouco sensíveis a erros nos dados ou arredondamentos numéricos quando o algoritmo é implementado no computador.

O problema de otimização pode ser escrito como:

$$\begin{array}{ll} \text{min} f(x) \quad \text{sujeito a} & g_k(x) \leq 0 \text{ para } k = 1, 2, \cdots, r \\ x \in \mathbb{R}^n & h_l(x) = 0 \text{ para } l = 1, 2, \cdots, s \end{array} \tag{2.1}$$

sendo que,

- x é o vetor das variáveis de projeto (ou variáveis de decisão, ou parâmetros do projeto);
- f é a função objetivo (ou índice de performance, ou função custo);
- g_k e h_l são funções de restrição (funções escalares de x que definem certas inequações ou equações, respectivamente, que as variáveis devem obedecer).

Existem diversos métodos de otimização e desta forma vários autores já apresentaram diferentes propostas para classificá-los, não sendo possível escolher uma proposta universalmente aceita. Os métodos dependem do tipo do problema em estudo (programação linear ou não-linear, problemas restritos ou irrestritos), das variáveis de projeto (contínuas ou discretas), do tipo de resposta desejada (otimização local ou global;

projeto ótimo ou controle ótimo), das técnicas empregadas (determinísticas, estocástica ou híbrida). Além disso, os problemas podem ser classificados segundo o número de variáveis (pequenos ou grandes) ou segundo a suavidade das funções (diferenciáveis ou não-diferenciáveis). A seguir serão apresentadas as denominações mais comuns nesta área:

- *Problemas de progamação linear ou não-lineares:* quando a função objetivo e todas as restrições são funções lineares de x (este tipo de problema é bastante comum em aplicações econômicas e nas áreas de transporte) denominam-se os problemas de progamação linear.

  Os problemas de progamação não-lineares são aqueles onde ao menos uma das restrições ou a função objetivo são não-lineares, eles aparecem naturalmente nas ciências físicas e nas engenharias.

- *Otimização irrestrita ou restrita:* problemas de otimização irrestritos são aqueles onde $I = D = \emptyset$. Este caso pode acontecer quando: os problemas práticos não dependem de restrições; as restrições foram desprezadas porque não afetam diretamente a solução ou o problema com restrição foi reescrito de forma que as restrições foram substituídas por funções de penalidade.

  Os problemas de otimização restritos, que aparecem devido às imposições do projeto, são aqueles onde $I \neq \emptyset$ e $D \neq \emptyset$. As restrições podem ser lineares (ex: $\sum x_i \leq 1$), laterais (ex: $0 \leq x_i \leq 20$), não-lineares (ex: $\sum x_i^2 - sen x_i \leq 0,5$), ou, ainda, combinações destas.

- *Otimização discreta ou contínua:* em alguns problemas de otimização as variáveis de projeto só têm sentido se forem considerados seus valores inteiros (ex: a variável x_i representa o número de caminhões em uma frota de transporte de cargas). Nestes tipos de problemas, deve-se incluir as "restrições de integralidade", ou seja $x_i \in \mathbb{Z}$ ($\mathbb{Z}$ é o conjunto dos números inteiros). Pode-se também incluir "restrições binárias", neste caso, $x_i \in \{0, 1\}$. Estes tipos de problemas é denominado problemas de otimização inteira ou *problemas de otimização discreta*. Neste caso, os

problemas podem considerar além das variáveis inteiras ou binárias, variáveis abstratas, as quais são representadas por conjuntos finitos (que podem ser bastante grandes). Na *otimização contínua* as variáveis x_i assumem valores no conjunto dos números reais.

- *Otimização global ou local:* a solução *global* é aquele ponto onde a função objetivo atinge o *menor* valor entre todos os pontos da região viável. A solução ou *ótimo local* é o ponto cuja função objetivo é menor que todos os pontos vizinhos da região viável, conforme representado na Fig. 2.1. Em muitas aplicações a solução global é desejada, mas difícil de ser reconhecida. Para problemas *convexos* e na *progamação linear* pode-se garantir que uma solução local é também solução global.

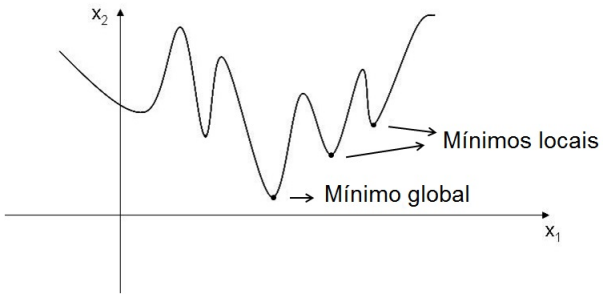


Figura 2.1 – Representação de mínimos locais e global

- *Projeto ótimo ou controle ótimo:* segundo ARORA (1987) o projeto ótimo e o controle ótimo de sistemas são duas atividades distintas. Existem numerosas aplicações onde a obtenção de projeto ótimo é útil para a concepção dos sistemas. Existem outras aplicações onde os conceitos de controle ótimo são necessários. Além disso, existem algumas aplicações onde tanto os conceitos de projeto ótimo quanto os de controle ótimo são usados. Uma amostra de onde isto ocorre inclui a robótica e estruturas aeroespaciais. Acontece que problemas de controle ótimo podem ser transformados em problemas de projeto ótimo e tratados pelos métodos de otimização. Um problema de controle ótimo visa encontrar uma entrada controlada para o sistema produzir a saída desejada. Neste caso, o sistema tem componentes ativos que percebem as flutuações na saída. Os sistemas de controles são automaticamente ajustados para corrigir a saída e otimizar alguma medida

de desempenho. Normalmente, o controle é aplicado em problemas de natureza dinâmica. Por outro lado, no projeto ótimo escreve-se o sistema e os componentes para otimizar a função objetivo. O sistema então permanece fixo durante toda a aplicação. Esta é a maior diferença entre as duas aplicações. Como um exemplo, considere o mecanismo de controle de viagem de um carro de passageiros. A idéia da resposta do sistema é controlar a injeção de combustível para manter a velocidade do carro constante. Assim, a saída do sistema é conhecida, ou seja, a velocidade da viagem. O trabalho do mecanismo de controle é perceber as flutuações na velocidade e ajustar adequadamente a injeção de combustível. A quantidade de injeção de combustível depende das condições da estrada. Quando o carro está numa subida, a injeção de combustível é maior do que em uma descida.

- *Otimização determinística ou estocástica:* nos problemas de *otimização determinística* o modelo é completamente conhecido. Os *algoritmos clássicos* de otimização, que dependem do conhecimento das derivadas da função objetivo, são exemplos de algoritmos determinísticos. Os melhores resultados destes métodos são para funções contínuas, convexas e semi-modais. Os métodos determinísticos possuem uma grande vantagem que é o baixo número de avaliações da função objetivo, fazendo com que tenham convergência rápida e baixo custo computacional.

  Na *otimização estocástica* pode-se trabalhar com "incertezas" de algumas variáveis e produzir soluções que otimizam a "performance esperada" do modelo. Pode-se garantir que as variáveis x satisfaçam às restrições para alguma probabilidade especificada anteriormente (modelos probabilísticos). Estes métodos também são conhecidos como métodos heurísticos. O termo heurístico tem sua origem na palavra grega "heuriskein" que significa "encontrar/descobrir". Estes métodos utilizam apenas informações da função a ser otimizada, que pode ser de difícil representação, não-diferenciável, descontínua, não-linear, multimodal. Alguns exemplos dos métodos heurísticos são os métodos naturais, que tentam simular os processos usados na natureza para resolver problemas complexos. Entre estes métodos pode-se citar: Busca Tabu, Simulated Annealing, métodos baseados em população

(Algoritmos Genéticos, Evolução Diferencial, Otimização por Colônia de Formigas, Otimização por Bando de Pássaros, etc). Estas técnicas trabalham com um conjunto de pontos, necessitam de muitas avaliações da função objetivo e sua maior desvantagem é o alto custo computacional.

- *A otimização híbrida:* consiste na utilização conjunta de métodos determinísticos e heurísticos. Tem sido largamente utilizada nos dias atuais, visa unir as boas características de convergência dos métodos determinísticos às vantagens dos métodos heurísticos, por exemplo, a independência do tipo de função e o fato de não ficarem "presos" pelos mínimos locais.

2.1 Otimização Multiobjetivo

Vários problemas das engenharias e de outras áreas possuem múltiplos objetivos a serem atingidos e nestes casos os modelos matemáticos não são mais representados apenas por uma única função, mas sim por várias funções objetivo simultaneamente. Esses problemas são conhecidos como problemas de otimização multiobjetivo, como problemas de tomada de decisão multicritério ou ainda por problema de otimização vetorial.

A intuição pode levar a acreditar que para se minimizar ou maximizar um problema multiobjetivo bastaria minimizar ou maximizar cada uma das funções separadamente, mas nem sempre a solução ótima de uma função representa o ponto ótimo para as outras funções. Visto que os objetivos são geralmente conflitantes entre si, ou seja, a melhora de um objetivo consiste na piora de um outro, no problema multiobjetivo podem ser encontrados não apenas uma única solução ótima, mas um conjunto de soluções favoráveis a todos os objetivos. Por isso, na otimização multiobjetivo a solução esperada é composta por um conjunto de soluções viáveis.

Assim, a modelagem em otimização multiobjetivo é formulada de modo a se encontrar um vetor de variáveis de projeto, o qual é representado por um vetor coluna de n variáveis $x = [x_1 x_2 \cdots x_n]^T \in \mathbb{R}^n$, que satisfará tanto as restrições do problema quanto otimizará um vetor de funções objetivos. As restrições do problema são dependências

entre variáveis e parâmetros intrínsecas ao problema que podem ser escritas na forma de igualdades e desigualdades matemáticas.

Matematicamente, a formulação de um problema multiobjetivo é dada da seguinte forma:

$$\begin{aligned} &\text{min ou max} \quad f(x) = [f_1(x) f_2(x) \cdots f_m(x)]^T \\ &\\ &\text{sujeito a} \qquad g_k(x) \leq 0 \ \text{ para } \ k = 1, 2, \cdots, r \\ &\qquad\qquad\ \ h_l(x) = 0 \ \text{ para } \ l = 1, 2, \cdots, s \\ &\qquad\qquad\ \ x_i^{inf} \leq x_i \leq x_i^{sup} \ \ i = 1, \cdots, n \end{aligned} \tag{2.2}$$

sendo que $f : \mathbb{R}^n \to \mathbb{R}^m$ é o vetor coluna de funções objetivos do problema onde cada $f_j : \mathbb{R}^n \to \mathbb{R}, \ j = 1, \cdots, m$, são funções uniobjetivo, g_k são as restrições de desigualdade, h_l as restrições de igualdade e x_i^{inf} e x_i^{sup} as restrições laterais de cada variável do problema.

Segundo OLIVEIRA (2005), o número s de restrições de igualdade deve ser menor que o número n de variáveis de projeto. Se $s \geq n$, o problema é denominado super restrito, uma vez que não existem graus de liberdade negativos na otimização. O número de graus de liberdade é dado por $n - s$.

Como cada ponto x no espaço vetorial $\mathbb{R}^n$ corresponde a um ponto $f(x)$ no espaço vetorial $\mathbb{R}^m$, a região viável X das variáveis de projeto determina uma outra região Y em $\mathbb{R}^m$ denominada espaço das funções objetivos.

Na otimização multiobjetivo pode-se procurar minimizar ou maximizar todas as funções f_j ou minimizar algumas e maximizar outras. Contudo, de agora em diante será admitido que se pretende minimizar todas as funções uniobjetivo, caso pretenda-se maximizar alguma ou todas as funções uniobjetivo esta ou estas serão substituídas pela função simétrica. Por exemplo, caso se queira maximizar uma função $f_j(x)$ qualquer, isto será feito por minimizar $-f_j(x)$.

Para problemas multiobjetivo mais simples onde as variáveis de projeto pertencem à $\mathbb{R}$ ou à $\mathbb{R}^2$ pode-se fazer uma análise gráfica por meio da qual encontra-se a solução ótima, porém, quanto maior a complexidade do problema mais difícil se torna

sua ilustração.

Há muitos métodos para se otimizar funções multiobjetivos. Alguns deste serão abordados a seguir.

2.1.1 Solução Ideal

Suponha que se possa encontrar a solução ótima, o ponto mínimo, de todas as funções objetivos f_j e seja este ponto ótimo denotado por $x^0 = [x_1^0 x_2^0 \cdots x_n^0]^T$. Assim, $f_i^0 = f_i(x^0) = min f_i(x),\ i = 1, 2, \cdots, m,\ \forall x \in \mathbb{R}^n$. Pode ser que x^0 não seja uma solução viável, mas o seu valor será bastante útil em alguns métodos de otimização. O ponto x^0 é chamado de solução ideal e o vetor $f^0 = [f_1^0 f_2^0 \cdots f_m^0]^T$ é chamado de vetor objetivo ideal.

Na maioria dos casos, o vetor objetivo ideal pode corresponder a uma solução que não existe já que as funções objetivos são conflitantes entre si, ou seja, as soluções ótimas para cada uma das funções objetivos podem ser diferentes de forma que otimizar uma função pode afetar as outras funções. Segundo OLIVEIRA (2005) a única maneira de um vetor ideal corresponder à solução viável seria se as funções objetivo fossem linearmente dependentes. Neste caso, os objetivos não seriam conflitantes uns com os outros e a solução ótima para qualquer uma das funções objetivos seria também a solução ótima para o problema de otimização multiobjetivo.

2.1.2 Ótimo de Pareto

Além das soluções ideais, pode-se obter várias outras possíveis soluções para o problema, por isso torna-se necessário formular uma comparação para as soluções obtidas a fim de selecionar aquelas que fornecem valores aceitáveis para todas as soluções objetivo. Num problema de otimização multiobjetivo há dois possíveis tipos de soluções: soluções que, quando comparadas às demais, apresentam pior desempenho sob todos os objetivos simultaneamente considerados e por esta razão este grupo deve ser descartado e soluções que, quando comparadas às demais, são melhores em um ou mais objetivos e que portanto, ao menos inicialmente, não devem ser descartadas.

O conjunto formado pelas soluções pertencentes a este último grupo, também

chamadas soluções eficientes ou soluções Pareto ótimas, é denominado conjunto Pareto ótimo. Um dos problemas centrais da otimização multiobjetivo é a determinação, completa ou parcial, do conjunto Pareto ótimo.

O Ótimo de Pareto é um conceito desenvolvido por Vilfredo Pareto, renomado economista italiano (1848-1923) e pioneiro em otimização multiobjetivo. Diz-se que uma situação econômica é ótima no sentido de Pareto se não for possível melhorá-la. Assim, tem-se um Ótimo de Pareto se, e somente se, nenhum agente ou situação estiver em uma posição melhor sem fazer com que outro agente ou situação assuma uma posição pior.

Afim de se obter o conjunto Pareto ótimo, que irá conter as possíveis soluções x^* do problema de otimização multiobjetivo (POM) serão definidos os elementos desse conjunto. O conceito de dominância será apresentado na definição 2.3. No entanto, faz-se necessário estabelecer generalizações das relações de ordem de menor e de menor ou igual no caso do conjunto $\mathbb{R}^n$ com $n \geq 2$.

Definição 2.1 *As operações de comparação entre vetores pertencentes ao espaço $\mathbb{R}^n$ são definidas da seguinte forma:*

$$\begin{aligned} x \leq y &\rightarrow \{x_i \leq y_i, \ \forall i = 1, \cdots, n\} \\ x < y &\rightarrow \{x_i < y_i, \ \forall i = 1, \cdots, n\} \\ x = y &\rightarrow \{x_i = y_i, \ \forall i = 1, \cdots, n\} \end{aligned}$$

Definição 2.2 *Diz-se que um conjunto A é parcialmente ordenado com respeito à relação de ordem $\leq$ se valem as seguintes propriedades*

$$\begin{aligned} &(i)\ x \leq x && \textit{(reflexividade)} \\ &(ii)\ x \leq y \textit{ e } y \leq z \rightarrow x \leq z && \textit{(transitividade)} \\ &(ii)\ x \leq y \textit{ e } y \leq x \rightarrow x = y && \textit{(antisimetria)} \end{aligned}$$

mas nem sempre $x \leq y$ ou $y \leq x$, isto é, nem sempre x e y são comparáveis.

Pode-se verificar que a relação $\leq$ assim definida estabelece um ordenamento parcial do conjunto $\mathbb{R}^n$, pois as propriedades de reflexividade, transitividade e antisimetria

de $\leq$ são facilmente verificáveis, enquanto que nem sempre será verdade que $x \leq y$ ou $y \leq x$, no caso de vetores x e y pertencentes ao $\mathbb{R}^n$.

Definição 2.3 *Diz-se que o ponto $x \in \mathbb{R}^n$ domina o ponto $y \in \mathbb{R}^n$ se $f(x) \leq f(y)$ e $f(x) \neq f(y)$. Equivalentemente, diz-se que $f(x) \in \mathbb{R}^m$ domina $f(y) \in \mathbb{R}^m$, nessas mesmas condições.*

O conceito de dominância para $\mathbb{R}^2$ encontra-se ilustrado na Fig. 2.2, na qual mostra-se os pontos y_A, y_B, y_C, y_D pertencentes ao espaço de objetivos Y. Para encontrar os pontos dominados ou dominantes basta traçar eixos paralelos aos eixos coordenados do espaço Y com vértices em cada ponto desse espaço, todos os pontos que estiverem no interior desse cone aberto são dominados pelo seu vértice. Assim, y_B e y_D dominam y_A e y_C, por outro lado, entre y_B e y_D não há relação de dominância.

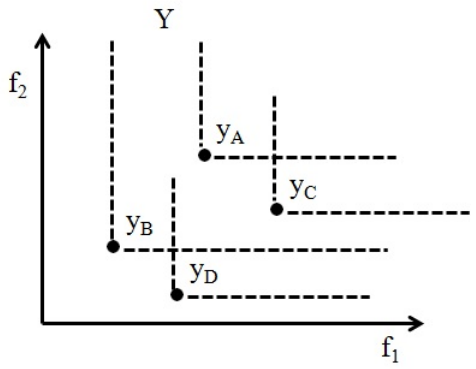


Figura 2.2 – Ilustração do conceito de dominância em $\mathbb{R}^2$. Fonte: TAKAHASHI (2007).

Seja S_x a região viável do espaço de parâmetros e S_y a região viável do conjunto imagem pela função f.

Definição 2.4 *Diz-se que $x^* \in S_x$ é uma solução Pareto Ótima do POM se não existe qualquer outra solução $x \in S_x$ tal que $f(x) \leq f(x^*)$ e $f(x) \neq f(x^*)$, ou seja, se x^* não é dominado por nenhum outro ponto viável.*

Este ponto ótimo forma um conjunto de soluções não inferiores, isto é, soluções para as quais não existem maneiras de melhorar algum critério sem piorar pelo menos um outro critério. É desta curva (ótimo de Pareto) que será selecionada a melhor solução para o problema.

Usa-se X^p para denotar este conjunto de soluções e F^p para denotar a região da imagem de X^p pela função f no espaço das funções objetivos.

Há muitos métodos que podem ser usados para minimizar uma função multiobjetivo por meio da obtenção da solução ótima de Pareto ou de um conjunto de tais soluções. Depois de encontrado este conjunto o usuário escolherá a melhor solução baseado nas informações que já dispõem a respeito da função multiobjetivo e das suas restrições. Alguns desses métodos serão apresentados em seguida.

2.1.3 Alguns métodos de otimização multiobjetivo

Nos métodos que serão estudados neste trabalho, o conjunto de funções é substituído por uma única função escalar que representa todas as funções objetivo. Assim, a obtenção da solução ótima se reduz a minimizar uma única função. A análise do vetor solução é realizada modificando a função principal de acordo com a importância de cada objetivo. Portanto um conjunto de soluções é selecionado de acordo com as modificações efetuadas na função principal, e cabe ao pesquisador analisá-las para escolher a melhor solução dentre elas.

Método da Ponderação dos Objetivos

Neste método, os problemas de otimização multiobjetivos são reformulados e substituídos por um único problema de otimização escalar. A idéia principal é criar uma única função principal a partir das funções objetivo, dando a elas certo grau de importância, isto é, ponderando seus valores em função da sua importância.

Esta manipulação algébrica das funções é dada da forma:

$$f(x) = \sum_{i=1}^{m} w_i f_i(x) \tag{2.3}$$

onde $w_i \geq 0$ são os coeficientes de ponderação, que representam a importância relativa

de cada critério. Deve-se assumir que:

$$\sum_{i=1}^{m} w_i = 1 \tag{2.4}$$

Levando em consideração que os resultados obtidos na resolução de um problema utilizando a modelagem descrita na Eq. (2.3) podem variar significamente com a mudança dos valores dos coeficientes de ponderação e que ainda sabe-se pouco a respeito de como se escolher estes coeficientes torna-se necessário obter diferentes aproximações variando os valores de w_i para resolver o mesmo problema. As quais, depois de obtidas, são comparadas para se escolher a melhor entre elas. Esta escolha é feita por meio da experiência do usuário e conhecimento do problema.

Para qualquer valor atribuído a w_i, os pontos obtidos estarão localizados na curva de Pareto, e a variação dos valores dos coeficientes pode ser um fator para a determinação do ótimo de Pareto.

Observe que a busca pelo ponto ótimo usando a Eq. (2.3) não depende apenas dos valores de w_i mas também das unidades nas quais as funções são expressas. Por exemplo, num problema em $\mathbb{R}^2$ pode ser que a escala de $f_1(x)$ seja da ordem de 10^6 enquanto que a escala de $f_2(x)$ seja da ordem de 10^{-4}. Assim, a variação dos parâmetros não causará grandes variações na solução, pois esta ficará próxima da solução ideal da primeira função f_1^0.

Dessa forma, para que w_i possa dar uma ideia da importância dos objetivos todas as funções devem ser expressas de forma adimencional. Esta técnica é aplicada utilizando os valores da solução ideal de cada função f_i^0 da seguinte forma:

$$f(x) = \sum_{i=1}^{m} w_i f_i(x) c_i \tag{2.5}$$

sendo c_i constantes multiplicadoras.

Assumindo que $f_i^0 \neq 0$, pode-se adotar $c_i = \frac{1}{f_i^0}$. Segundo OLIVEIRA (2005), esse valor tem apresentado os melhores resultados. Caso $f_i^0 = 0$, outro valor pode ser escolhido pelo usuário.

O método da ponderação dos objetivos é mais indicado quando se deseja priorizar um determinado critério e por este motivo o resultado será fortemente influenciado pela escolha do critério mais importante.

Método do Critério Global

O método do critério global utiliza um valor estabelecido como ideal para cada função como base de cálculo para definir o grau de importância de cada ponto x da região viável dos parâmetros. Assim, a solução ótima será um vetor de variáveis de decisão o qual minimiza algum critério global. Esse método converte a função multiobjetivo em um único objetivo sendo expresso matematicamente pela seguinte função:

$$f(x) = \sum_{i=1}^{m} \left(\frac{f_i^0 - f_i(x)}{f_i^0} \right)^s \tag{2.6}$$

sendo que usualmente usa-se $s = 1$ ou $s = 2$, mas outros valores podem ser adotados.

É claro que para cada valor de s a solução obtida minimizando a Eq. (2.6) será diferente. Por isso, o problema é determinar qual valor de s que resultará na solução que seja a mais satisfatória para o pesquisador. Também, não se pode garantir a existência de soluções satisfatórias pois pode acontecer que para qualquer valor de s escolhido o método forneça uma solução inaceitável do ponto de vista do usuário.

A Eq. (2.6) não é o único modo de se formular a função de critério global para otimização multiobjetivo, outra maneira possível é por meio de uma família de Métricas L_p definida como:

$$L_p(f) = \left(\sum_{i=1}^{m} \left| f_i^0 - f_i(x) \right|^s \right)^{1/s}, \quad 1 \leq s \leq \infty \tag{2.7}$$

Se $s = 1$ a métrica se resume em $L_1(f) = \sum_{i=1}^{n} |f_i^0 - f_i(x)|$. Por outro lado, $L_\infty(f) = max|f_i^0 - f_i(x)|$. As métricas mais utilizadas são $L_1(f), L_2(f)$ e $L_3(f)$. Observe que, a minimização de $L_2(f)$ equivale à minimização da distância Euclidiana entre o valor da função e a solução ideal.

Há ainda outra técnica na qual em vez de se trabalhar com a distância no

sentido absoluto, utiliza-se o valor das distâncias relativas. Esta técnica é dada de forma geral por:

$$L_p(f) = \left(\sum_{i=1}^{m} \left| \frac{f_i^0 - f_i(x)}{f_i^0} \right|^s \right)^{1/s}, \quad 1 \leq s \leq \infty \tag{2.8}$$

Como existem muitas formulações de critérios globais é importante que se aplique muitas destas para que seja possível comparar as soluções encontradas para então escolher a melhor entre elas.

Os métodos globais são indicados para os casos onde se deseja obter uma solução que atenda todas as funções objetivos.

2.2 Otimização Restrita - Método da Penalidade

Muitos algoritmos de otimização foram desenvolvidos para resolver problemas irrestritos, considerando apenas os limites laterais das variáveis, como no caso do Algoritmo Genético e da Evolução Diferencial, enquanto apenas alguns algoritmos processam as restrições de desigualdade e igualdade do problema para limitar a região viável. Assim, torna-se necessário utilizar algum artifício para que os problemas com restrições também se apliquem aos métodos de otimização irrestrita.

Uma das abordagens fundamentais para a otimização restrita é substituir o problema original por uma função de penalidade que é composta da função objetivo original do problema de otimização com restrições somada a um termo adicional para cada restrição, o qual é positivo quando o ponto atual X viola essa restrição e zero caso contrário.

A maioria das abordagens definem uma sequência de funções de penalidade, na qual os termos penalizados por violarem as restrições são multiplicados por um coeficiente positivo. Ao fazer este coeficiente crescer, penalizam-se as violações das restrições mais severamente, forçando o ponto mínimo da função de penalidade se aproximar cada vez mais da região viável do problema restrito.

Essas abordagens são conhecidas como Método da Penalidade Exterior, porque o termo penalizado por cada restrição é diferente de zero somente quando X é inviável em relação a tal restrição. Segundo VANDERPLAATS (1984), muitas vezes, o ponto

de mínimo das funções de penalidade são inviáveis com relação ao problema original, e aproximam-se da viabilidade apenas no limite em que os parâmetros de penalidade se tornam cada vez maiores.

Considere o seguinte problema:

$$\begin{array}{lll} \text{min } f(X) & \text{sujeito a} & g_j(X) \leq 0, \quad \text{restrições de desigualdade} \\ X \in \mathbb{R}^n & & h_l(X) = 0, \quad \text{restrições de igualdade} \end{array} \tag{2.9}$$

Afim de que os problemas restritos (2.9) sejam transformados em problemas irrestritos será utilizado, neste estudo, o Método da Penalidade Exterior. Se o objetivo da otimização é minimizar a função objetivo $f(X)$, a função de penalidade $P(X)$ é somada a função principal $f(X)$ de modo a aumentar o valor da função nos pontos que estão fora da região viável, por outro lado, se o objetivo é maximizar, a função de penalidade é subtraída da função principal se o ponto não obedece às restrições.

Esta nova função objetivo, chamada *pseudo objetivo*, é penalizada de acordo com um *fator de penalidade* r_p toda vez que encontrar uma restrição ativa. Assim, este escalar amplia a penalidade, e quanto maior for seu valor, maior será a eficiência do método para obedecer às restrições. O modelo matemático da função pseudo objetivo $\Phi(X)$ utilizada para minimizar uma função $f(X)$ é dada por:

$$\Phi(X) = f(X) + r_p P(X), \tag{2.10}$$

e a função de penalidade $P(X)$ dada por:

$$P(X) = \sum_{j=1}^{m} \left(max\left[0, g_j(X)\right]^2 \right) + \sum_{k=1}^{l} h_l(X)^2, \tag{2.11}$$

sendo $g_j(X)$ as restrições de desigualdade, $h_l(X)$ as restrições de igualdade e m e l o número de restrições de desigualdade e igualdade respectivamente.

Deste modo, os pontos fora do espaço viável apresentarão valores muito ruins para a otimização e serão facilmente desconsiderados pelos algoritmos de otimização irrestrita.

CAPÍTULO III

ESTUDO DE ALGUNS MÉTODOS EVOLUTIVOS

3.1 Introdução aos Processos Evolutivos

A tarefa de imitar as estruturas e processos biológicos com o objetivo de resolver problemas técnicos é tão antiga quanto a engenharia. O mito de Dédalo e Ícaro comprova este empreendedorismo humano.

A estratégia evolutiva (EE) teve origem em 1964 na Universidade Técnica de Berlim, Alemanha. O problema original em estudo era o da otimização de forma para objetos inseridos em túneis de vento. Estratégias usando o método do gradiente não foram bem sucedidas. Dois estudantes, Ingo Rechenberg e Hans-Paul Schwefel tiveram a ideia de efetuar alterações randômicas nos parâmetros que definiam a forma dos objetos, baseados na idéia da seleção natural. Este estudo resultou no mecanismo denominado $(1+1)$, um esquema simples de seleção-mutação, que trabalha em um único indivíduo, que gera um único descendente por geração, através da mutação Gaussiana (BÄCK; HOFFMEISTER; SCHWEFEL (1991)). Mais tarde, esta teoria evoluiu para o chamado mecanismo $(\mu+1)$. Neste caso, uma população de μ indivíduos se recombina de maneira randômica para formar um descendente, o qual, após sofrer mutação, substitui (se for o caso) o pior elemento da população. Ainda que este mecanismo não tenha sido muito usado, ele permitiu a transição para os mecanismos chamados $(\mu+\lambda)$ e (μ,λ), já no final

dos anos 70. Na primeira, os μ ancestrais e os λ descendentes convivem, enquanto na segunda, os μ ancestrais "morrem", deixando apenas os λ descendentes "vivos".

A regra de cada iteração k para minimizar uma função por meio de uma busca aleatória é:

$$x^{k+1} = \begin{cases} x^k + z^k & \text{se } f(x^k + z^k) \leq f(x^k) \quad \text{sucesso} \\ x^k & \text{caso contrário} \qquad\quad \text{falha} \end{cases} \tag{3.1}$$

O vetor aleatório z^k, que nesta notação efetua mudanças no vetor x^k, pertence à distribuição normal n-dimensional $(0, \sigma^2)$ com esperança $\xi = 0$ e variância σ^2, que no caso mais simples é a mesma para todas as componentes. Pode-se, assim, considerar σ, ou melhor $\sigma\sqrt{n}$, como um tipo de comprimento do passo médio. A direção de z^k é uniformemente distribuída em $\mathbb{R}^n$, isto é, sua escolha é puramente aleatória. Distribuições Gaussianas para os incrementos também são utilizados por BEKEY et al (1966), STEWART; KAVANAUGH; BROCKER (1967) e DEGRAAG (1970). Os autores GONZALEZ (1970) e WHITE (1970) adotaram, ao invés da distribuição normal, a distribuição uniforme, que considera uma pequena região na forma de um cubo n-dimensional centrado no ponto inicial.

Segundo HEITKOETTER e BEASLEY (1994), a abordagem EE é adequada para uma vasta gama de problemas de otimização, porque não necessita de muitas informações sobre o problema. Ela é capaz de resolver problemas multidimensionais, multimodais e não lineares sujeitos a restrições lineares ou não lineares.

Na estratégia evolucionária, um indivíduo corresponde a um ponto no espaço de soluções, e possui um genótipo formado por variáveis objetivos e variáveis estratégicas. As variáveis objetivo são aquelas que, sofrendo recombinação e mutação, permitem incrementar a aptidão dos indivíduos em direção ao mínimo ou máximo global de otimização. As variáveis estratégicas representam variâncias e co-variâncias, que devem ser operadas junto às variáveis de controle para produzir mutações.

Existem vários tipos de estratégias evolutivas, podendo ser citadas: Dois-membros: $(1+1)$-EE, Multimembros: $(\mu+1)$-EE, Multimembros: $(\mu+\lambda)$-EE, Multimembros: (μ, λ)-EE.

Por exemplo, na estratégia $(\mu+\lambda)$-EE tem-se que de $(\mu+\lambda)$ indivíduos, μ serão

selecionados, já na estratégia (μ, λ)-EE de λ indivíduos, μ serão selecionados $(\lambda \geq \mu)$. Nas próximas seções serão discutidas algumas estratégias evolutivas.

3.1.1 Estratégia (1+1)

Na estratégia (1+1) uma solução gera outra a cada geração. Nessa operação é aplicada uma mutação normal (ou seja, pequenas alterações têm maior probabilidade de ocorrer do que grandes alterações, seguindo a distribuição normal) até que o descendente tenha um desempenho melhor que seu ascendente, a partir de então ele toma o lugar. Assim, o filho (indivíduo mutado) é aceito na nova geração se, e somente se, ele possuir um custo melhor do que o do pai (se for viável).

Nos termos da biologia, as regras, segundo SCHWEFEL (1995), são as seguintes:

1. **Inicialização:** Uma dada população é constituída por dois indivíduos, um pai e um descendente (filho). Cada um é identificado por seu genótipo de acordo com um conjunto de n genes. Apenas o genótipo parental tem de ser especificado como ponto de partida.

2. **Mutação:** O pai p^q da geração q produz um descendente d^q, cujo genótipo é ligeiramente diferente do seu. As variações se referem aos genes individuais, sendo aleatórios e independentes uns dos outros.

3. **Seleção:** Por causa da diferença dos genótipos, os dois indivíduos têm uma capacidade diferente para a sobrevivência (no mesmo ambiente). Apenas um dos dois (pai ou descendente) pode produzir descendentes na próxima geração. O indivíduo que tiver a maior capacidade de sobrevivência se tornará o pai p^{q+1} da próxima geração $q + 1$.

Assim, algumas suposições podem ser feitas:

- O tamanho da população permanece constante;

- Um indivíduo tem, em princípio, um tempo infinitamente longo de vida e capacidade para produzir descendentes (assexuadamente);

- Não existem diferenças entre genótipo (codificação) e fenótipo (aparência), ou esta é inequívoca e reproduzível associada com o outro;
- Somente pontos de mutação ocorrem, independentemente de cada um de todos os parâmetros locais;
- O ambiente e, portanto, o critério de sobrevivência é constante ao longo do tempo.

No algoritmo, dado a seguir, os índices p representa pai, d representa descendente (filho) e q representa a q-ésima geração:

$$\begin{cases} \text{min } \ f(X), & X \in \mathbb{R}^n \\ \text{sujeito a } g_j(X) \leq 0, & j = 1, \cdots, m \end{cases} \tag{3.2}$$

Passo 1: (Inicialização) Defina X_p^0 tal que $g_j(X^0) \leq 0$ para todo $j = 1, \cdots, m$.

Passo 2: (Mutação) Faça $X_d^q = X_p^q + z^q$ com componentes $X_{d,i}^q = X_{p,i}^q + z_i^q$ para todo $i = 1, \cdots, n$.

Passo 3: (Seleção) Escolha

$$X_p^{q+1} = \begin{cases} X_d^q, & \text{se } f(X_d^q) < f(X_p^q) \text{ e } g_j(X_d^q) \leq 0 \text{ para todo } j = 1, \cdots, m \\ \\ X_p^q, & \text{caso contrário} \end{cases}$$

Faça $q = q + 1$ e vá para o passo 1 até que algum critério de parada seja satisfeito.

Apesar de existir um único indivíduo na população, este procedimento é denominado de estratégia evolutiva de dois membros pois o filho compete com o pai. Sendo assim, durante o processo de seleção (competição) existem, temporariamente, dois indivíduos na população.

A questão que permanece é como escolher os vetores aleatórios z^q, denominados fatores de mutação. Esta escolha deve obedecer à regra de mutação. As mutações, como são entendidas hoje em dia, devem ser aleatórias, eventos sem finalidade (*purposeless events*), que, além disso, só ocorrem muito raramente. Se interpretado como uma soma de muitos eventos individuais (como é feito aqui), é natural escolher uma distribuição de probabilidade de acordo com a observação biológica de que pequenas variações ocorrem com maior frequência do que grandes variações, e de que os filhos herdam características dos pais, ou seja, são parecidos com os pais (teorema do limite central das estatísticas).

Para variações discretas, pode-se usar a *distribuição binomial*, por exemplo, e para variações contínuas a *distribuição Gaussiana* ou *distribuição normal*.

Duas exigências surgem por analogia com a evolução natural:

- Que o valor da esperança ξ_i para cada componente z_i vale zero;
- Que a variância σ^2, o desvio médio quadrado para a média, seja pequena.

A função densidade de probabilidade para eventos aleatórios distribuídos normalmente vale:

$$w(z_i) = \frac{1}{\sigma_i\sqrt{2\pi}} exp\left(-\frac{(z_i - \xi_i)^2}{2\sigma_i^2}\right) \tag{3.3}$$

Se $\xi_i = 0$, obtém-se a chamada distribuição normal $(0, \sigma_i^2)$. Há ainda, contudo, um total de n parâmetros livres $\{\sigma_i, i = 1, \cdots, n\}$ para especificar os desvios-padrão dos componentes aleatórios individuais. Por analogia, nas estratégias de busca determinísticas, σ_i pode ser chamado tamanho do passo, no sentido de que eles representam valores médios dos comprimentos dos passos aleatórios.

Para a ocorrência de um vetor aleatório particular $z = \{z_i, i = 1, \cdots, n\}$, cujas componentes são distribuídas independentemente $(0, \sigma_i^2)$, a função densidade de probabilidade é dada por:

$$w(z) = w(z_1, \cdots, z_n) = \prod_{i=1}^{n} w(z_i) = \frac{1}{(2\pi)^{n/2}\prod_{i=1}^{n}\sigma_i} exp\left(-\frac{1}{2}\sum_{i=1}^{n}\left(\frac{z_i}{\sigma_i}\right)^2\right) \tag{3.4}$$

ou mais compactamente, se $\sigma_i = \sigma$ para todo $i = 1, \cdots, n$,

$$w(z) = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n exp\left(\frac{-zz^T}{2\sigma^2}\right) \tag{3.5}$$

O próprio Darwin, (DARWIN (1966)), salientou que a evolução dos seres vivos não é um processo puramente aleatório. No entanto, contra a sua teoria da descendência, a polêmica ainda está travada na impossibilidade de se demonstrar que a vida poderia surgir de um processo puramente aleatório. Mesmo ao nível da simples imitação da evolução orgânica, uma escolha adequada do tamanho do passo ou das variâncias são de fundamental significância.

3.1.2 Estratégias soma e vírgula

Note que até o momento o conceito de população ainda não foi utilizado. Para introduzir o conceito de população no algoritmo, Rechenberg propôs as EE's multimembros. As estratégias $(\mu + \lambda)$ e (μ, λ), são chamadas *estratégia soma* e *estratégia vírgula*, respectivamente. Na estratégia soma, os ascendentes são levados em conta durante a etapa de seleção, enquanto na estratégia vírgula, apenas os descendentes de uma dada geração são candidatos a serem selecionados para gerar a próxima. Esta característica dos ancestrais competirem junto com os descendentes frente ao operador seleção é chamada de elitismo. A escolha adequada de μ/λ determina a velocidade de convergência da EE (HEITKOETTER e BEASLEY (1994)).

Na estratégia $(\mu + \lambda)$-EE, μ pais produzem λ filhos e a população $\mu + \lambda$ é posteriormente reduzida para μ indivíduos. A seleção opera no conjunto união de pais e filhos. Assim, os pais sobrevivem até que filhos com aptidão superiores a eles sejam produzidos. Em problemas com funções objetivo dinâmicas (que variam ao longo do tempo), a estratégia $(\mu + \lambda)$-EE pode ficar presa em um ótimo que não é mais um ótimo da superfície atual. O mesmo pode ocorrer na presença de ruído. Para evitar estes efeitos, SCHWEFEL (1995) investigou as propriedades de uma estratégia (μ, λ)-EE, onde somente os filhos sofrem seleção.

Como ambas as estratégias $(\mu+\lambda)$-EE e (μ, λ)-EE possuem a mesma estrutura geral, será feito apenas uma análise das estratégias (μ, λ)-EE.

3.1.3 Estratégia Evolutiva (μ, λ)

Na estratégia evolutiva (μ, λ) uma população de μ pais produzem $\lambda > \mu$ descendentes, mas os μ pais não são incluídos na seleção. Pelo contrário, os pais da próxima geração são selecionados somente a partir dos λ descendentes. Para manter o tamanho da população constante, é requerido que os melhores μ dos λ descendentes se tornem pais da próxima geração, ou seja, o período de vida de cada indivíduo está restrito a *uma* geração. O período de vida restrito permite a eliminação de parâmetros inapropriados SCHWEFEL (1987). O algoritmo da EE multimembros (μ, λ) será inicialmente formulado em linguagem biológica:

1. Uma dada população consiste de μ indivíduos. Cada um é caracterizado pelo seu genótipo, consistindo de n genes, que determina de modo não ambíguo a aptidão para a sobrevivência.

2. Cada indivíduo da população produz λ/μ, $(\lambda \geq \mu)$ descendentes, na média, de modo que um total de λ indivíduos novos são gerados. O genótipo dos descendentes difere ligeiramente dos genótipos de seus ancestrais.

3. Apenas os μ melhores indivíduos dos λ gerados permanecem vivos, tornando-se os ancestrais na próxima geração.

A principal diferença entre as diversas variantes das estratégias evolutivas está na forma de atualização do vetor de parâmetros σ. Note que, embora anteriormente σ já fazia parte da codificação genética de cada indivíduo da população, seu valor era atualizado heuristicamente, e não evoluído como será daqui para diante.

Alguns autores (p.ex. BÄCK; HOFFMEISTER; SCHWEFEL (1991)) não assumem que σ faz parte da codificação dos indivíduos como foi apresentado aqui, trata-se apenas de um parâmetro externo. A partir de agora o operador de mutação vai atuar diretamente em σ.

Em seguida será apresentado o algoritmo em notação matemática, levando-se em conta as definições anteriores. Antes, porém, é necessário recordar a definição de congruência: diz-se que *a* é congruente a *b* módulo *m* se $m|(a-b)$, isto é, se m divide $a-b$. Notação: $a \equiv b(mod\ m)$.

Uma propriedade importante da congruência é que se $a \equiv b(mod\ m)$ então existe um inteiro k tal que $a = b + km$.

O algoritmo é dado a seguir:

Passo 1: (Inicialização)

Defina $x_h^0 = x_{ph}^0 = (x_{h,1}^0, \cdots, x_{h,n}^0)^T$ para todo $h = 1, \cdots, \mu$.
O vetor $x_h^0 = x_{ph}^0$ é o vetor do h-ésimo pai $x_{p,h}$, tal que $C_j(x_h^0) \geq 0$ para todo $h = 1, \cdots, \mu$ e todo $j = 1, \cdots, m$. Seja $q = 0$.

Passo 2: (Mutação)

Gere $x_l^{q+1} = x_h^{q+1} + z^{q\lambda+l}$,
tal que $C_j(x_l^{q+1}) \geq 0$, $j = 1, \cdots, m$, $l = 1, \cdots, \lambda$, onde

$$h = \begin{cases} \mu & \text{se } l = p\mu,\ p \text{ inteiro} \\ l(mod\mu) & \text{caso contrário} \end{cases}$$

O vetor $x_l^{q+1} = x_{d,l}^{q+1} = (x_{l,1}^{q+1}, \cdots, x_{l,n}^{q+1})^T$ é o vetor do l-ésimo descendente $x_{d,l}$,
e $z^{q\lambda+l}$ é um vetor randômico distribuido normalmente com n componentes.

Passo 3: (Seleção)

Selecione x_l^{q+1} para todo $l = 1, \cdots, \lambda$, de forma que
$f(x_{l_1}^{q+1}) \leq f(x_{l_2}^{q+1})$, para todo $l_1 = 1, \cdots, \mu$ e $l_2 = \mu + 1, \cdots, \lambda$
atribua $x_h^{q+2} = x_{l_1}^{q+1}$, para todo $h, l_1 = 1, \cdots, \mu$.
Faça $q = q + 1$ e vá para o passo 1 até que algum critério de parada seja satisfeito.

Além das estratégias evolutivas, existem muitos outros algoritmos baseados na seleção natural. Na realidade, as EE's foram o começo dos estudos dos métodos de otimização natural. Mas, quase ao mesmo tempo em que as EE's foram desenvolvidas e

utilizadas na Universidade Tecnológica de Berlin, duas outras linhas de algoritmos evolucionários (AE's) surgiram nos Estados Unidos, independentes entre si. Uma delas é a progamação linear e a outra são os algoritmos genéticos.

3.2 Algoritmos Genéticos

Charles Darwin, estudando as espécies e suas evoluções, coletou durante anos uma grande quantidade de material que comprovaram, principalmente, a existência de inúmeras variações em cada espécie. Seus estudos, associados às pesquisas de outros cientistas do assunto, tornaram evidentes que as espécies animais se modificam. Um dos principais pontos dos estudos de Darwin foi o aspecto das variações apresentadas entre indivíduos da mesma espécie. Através de estudos em pombos, por exemplo, ele observou a enorme variedade de indivíduos que se obtém, cruzando uma mesma espécie. Segundo esse estudioso, todas as novas espécies são produzidas por meio de uma seleção natural.

Há quase cinco bilhões de anos, a natureza vem resolvendo problemas com sucesso. Cada organismo possui cromossomos, genes, exons, íntrons e códons, constituindo um sistema genético. Um determinado grupo de indivíduos que vivem junto, constituem uma população. Nesta população existem organismos mais hábeis, que são os que têm maior chance de gerar bons descendentes. Estes descendentes são mais aptos do que a média da população, pois receberam genes melhores. Darwin mostrou que apenas os mais adaptados ao nicho ecológico da população irão sobreviver. Estes transmitirão suas características às gerações subsequentes, melhorando a cada geração a aptidão dos seus descendentes.

Por volta de 1900, o trabalho de Gregor Mendel, desenvolvido em 1865, sobre os princípios básicos de herança genética, foi redescoberto pelos cientistas e teve grande influência sobre os futuros trabalhos relacionados à evolução. A teoria moderna da evolução combina a genética e as ideias de Darwin e Wallace sobre a seleção natural, criando o princípio básico de Genética Populacional: a variabilidade entre indivíduos de uma população de organismos que se reproduzem sexualmente é produzida pela

mutação e pela recombinação genética.

Este princípio foi desenvolvido durante os anos 30 e 40, por biólogos e matemáticos de importantes centros de pesquisa. Nos anos 50 e 60, muitos biólogos começaram a desenvolver simulações computacionais de sistemas genéticos. Entretanto, foi John Holland quem começou, seriamente, a desenvolver as primeiras pesquisas no tema. Holland foi gradualmente refinando suas ideias e em 1975 publicou o seu livro Adaptation in Natural and Artificial Systems, hoje considerado a Bíblia dos Algoritmos Genéticos (HOLLAND (1975)). O método foi desenvolvido por Holland durante cursos nas décadas de 60 e 70, mas foi um dos seus alunos, David Goldberg, quem popularizou o método ao resolver um difícil problema envolvendo o controle de trasmissão de gás em uma tubulação, em sua dissertação em 1989 (GOLDBERG (1989)). Desde então, estes algoritmos vêm sendo aplicados com sucesso nos mais diversos problemas de otimização e aprendizado de máquina.

Segundo GOLDBERG (1989), os Algoritmos Genéticos (AGs) diferem dos métodos tradicionais de busca e otimização, principalmente em quatro aspectos:

1. AGs trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros.

2. AGs trabalham com uma população e não com um único ponto.

3. AGs utilizam informações de custo ou recompensa e não derivadas ou outro conhecimento auxiliar.

4. AGs utilizam regras de transição probabilísticas e não determinísticas.

Algoritmos Genéticos são algoritmos de otimização global, baseados nos mecanismos de seleção natural e da genética. Eles empregam uma estratégia de busca paralela e estruturada, mas aleatória, que é voltada em direção ao reforço da busca de pontos de “alta aptidão”, ou seja, pontos nos quais a função a ser minimizada (ou maximizada) tem valores relativamente baixos (ou altos).

Apesar de aleatórios, eles caminham de forma direcionada, pois exploram informações históricas para encontrar novos pontos de busca onde são esperados melhores

desempenhos. Isto é feito através de processos iterativos, onde cada iteração é chamada de geração.

Durante cada iteração, os princípios de seleção e reprodução são aplicados a uma população de candidatos que pode variar, dependendo da complexidade do problema e dos recursos computacionais disponíveis. Através da seleção, se determina quais indivíduos conseguirão se reproduzir, gerando um número determinado de descendentes para a próxima geração, com uma probabilidade determinada pelo seu índice de aptidão.

Inicialmente, é gerada uma população formada por um conjunto aleatório de indivíduos que podem ser vistos como possíveis soluções do problema. Durante o processo evolutivo, esta população é avaliada: para cada indivíduo é dada uma pontuação, ou índice, refletindo sua habilidade de adaptação a determinado ambiente. Uma porcentagem dos mais adaptados é mantidas, enquanto os outros indivíduos são descartados. Os membros mantidos pela seleção podem sofrer modificações em suas características fundamentais através de mutações e cruzamento ou recombinação genética gerando descendentes para a próxima geração. Este processo, chamado de reprodução, é repetido até que uma solução satisfatória seja encontrada.

Embora possam parecer simplistas do ponto de vista biológico, estes algoritmos são suficientemente complexos para fornecer mecanismos adaptativos de busca poderosos e robustos.

Para realizar o processo de otimização via Algoritmo Genético é necessário seguir seis operações básicas que compreendem:

1. codificação das variáveis do problema;
2. criação dos indivíduos iniciais, ou seja, formar a população inicial;
3. avaliar os indivíduos da população;
4. selecionar tais indivíduos, de acordo com critérios estabelecidos;
5. realizar o cruzamento entre os indivíduos e;
6. aplicar mutação nestes indivíduos.

As operações seleção, cruzamento e mutação são repetidas até atingir algum critério de parada. Finalmente, o melhor individuo é apresentado como a solução do problema.

Na terminilogia natural, cromossomos são compostos de genes, que podem tomar alguns números de valores chamados alelos. A posição de um gene (locus) é identificada separadamente a partir da função que o gene assume, desta forma, pode-se falar de um gene particular. Por exemplo, no gene que determina a cor dos olhos de um animal, o locus é a posição 10 e o valor do alelo, olhos azuis. Assim, gene é a caracteristica e o alelo é o valor da característica.

Como os algorítmos genéticos usam um vocabulário emprestado da genética natural, da-se a seguir uma breve descrição dos termos mais comuns:

- Cromossomo (genótipo): Estrutura que representa as variáveis do problema e é chamado de indivíduo, o qual forma a população do Algoritmo Genético. Cada cromossomo corresponde a uma solução do problema dentro do espaço de busca (um ponto do espaço).
- Gene: São os constituintes dos cromossomos, ou seja, cada cromossomo é formado por um conjunto de genes. Cada gene representa uma variável de projeto x_i.
- Alelos: São as unidades presentes nos genes, cada gene é composto por m alelos, donde m é o comprimento do gene.
- Fenótipo: Representa um cromossomo decifrado.
- Operações Genéticas: Operações realizadas sobre os cromossomos através de sucessivas iterações. Estas incluem seleção, cruzamento e mutação.
- Geração: Corresponde à iteração atual do Algoritmo Genético.

A Fig. 3.1 exemplifica estas definições.

O Algoritmo Genético clássico para otimização de parâmetros tem sido apresentado da seguinte forma SCHWEFEL (1995):

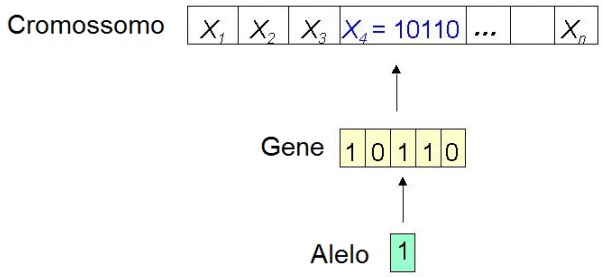


Figura 3.1 – Esquematização de um cromossomo. Fonte: OLIVEIRA (2006).

Passo 1: (Inicialização)

Uma dada população é composta por λ indivíduos. Cada um é caracterizado pelo seu genótipo composto por n genes, que determinam a vitalidade, ou adaptação para sobrevivência. O genótipo de cada indivíduo é representado por uma sequência de bits binários, representando os valores dos parâmetros objetivos diretamente ou por meio de um esquema de codificação.

Passo 2: (Seleção)

Dois pais são escolhidos com probabilidade proporcional à sua posição relativa na população atual, ou medidos por sua contribuição para o valor médio da função objetivo na geração (seleção proporcional) ou por sua posição (seleção ranking linear).

Passo 3: (Cruzamento)

Dois descendentes diferentes são produzidos pela recombinação de dois genótipos parentais por meio de cruzamento de uma dada probabilidade de recombinação p_c. Apenas um dos filhos é tomado aleatoriamente em consideração nesta etapa. Os passos 1 e 2 são repetidos até que λ indivíduos representem a próxima geração.

Passo 4: (Mutação)

Os descendentes com uma dada probabilidade p_m fixa e pequena sofrem outras alterações subjacentes por meio de mutações pontuais em bits individuais, quer invertendo 1 por 0, ou vice-versa, ou lançando um dado para a escolha de 0 ou 1, independentemente do valor original.

Além do algoritmo clássico apresentado, há também vários outros esquemas propostos por estudiosos para as operações de seleção, cruzamento e mutação e cada uma das operações podem ter diversas versões diferentes. Algumas modificações propostas no esquema do AG podem ser vistas em NETO et al (2005), LINS (2014), OCAMPO; GRISALES; ECHEVERRI (2006).

Um estudo detalhado de cada operador do AG clássico é apresentado em OLIVEIRA (2006) e BRANDÃO e SARAMAGO (2011).

3.3 Evolução Diferencial

A Evolução Diferencial (ED) foi criada após Ken Price tentar usar técnicas evolutivas para resolver o problema do polinômio de Chebychev, o qual foi apresentado por Rainer Storn. A descoberta aconteceu quando Price teve a ideia de utilizar diferentes vetores para recriar o vetor população criando uma abordagem diferente das encontradas nas estratégias de evolução que tratam de problemas de otimização. Desde então foram feitos vários testes e aperfeiçoamentos, os quais tornaram o algoritmo ED versátil e robusto.

O algoritmo começa criando uma população inicial de N_p indivíduos, escolhida aleatoriamente e devendo cobrir todo o espaço de busca. Geralmente, é criada por uma distribuição de probabilidade uniforme, quando não há nenhum conhecimento sobre o problema.

Cada indivíduo, chamado de *vetor*, possui n componentes representadas por valores reais, sendo n o número de variáveis de projeto. Assim, a população segue uma evolução natural, em que o número de indivíduos é constante durante todas as gerações.

A idéia principal da evolução diferencial é gerar novos indivíduos, denotados vetores modificados ou doadores, pela adição da diferença ponderada entre dois indivíduos aleatórios da população a um terceiro indivíduo. Esta operação é chamada *mutação*.

As componentes do indivíduo doador são misturadas com as componentes de um indivíduo escolhido aleatoriamente (denotado vetor alvo), para resultar o chamado

vetor tentativa, ou vetor experimental. O processo de misturar os parâmetros é referido como *cruzamento*. Se o vetor experimental resultar um valor da função objetivo menor que o vetor alvo, então o vetor experimental substitui o vetor alvo na geração seguinte. Esta última operação é chamada *seleção*. O procedimento é finalizado quando algum critério de parada é obedecido.

3.3.1 Operadores da Evolução Diferencial

Assim como o Algoritmo Genético, a Evolução Diferencial apresenta algumas operações a serem seguidas durante cada geração para manter a convergência não-prematura do método, garantindo, então, a diversidade da população e a obtenção da solução ótima, ou próxima desta.

Considere o seguinte problema de otimização sem restrições (a menos das laterais):

$$\begin{cases} \text{min } \ f(X), \quad X \in \mathbb{R}^n \end{cases} \tag{3.6}$$

onde $f(X)$ representa a função objetivo (também chamada de função custo ou adaptação), e sejam $X^L = (x_1^L, x_2^L, \cdots, x_n^L)^T$ e $X^U = (x_1^U, x_2^U, \cdots, x_n^U)^T$ as restrição laterais inferior e superior, respectivamente.

A população de indivíduos durante a q-ésima geração é definida como:

$$X_d^{(q)} = (x_{d,1}, x_{d,2}, \cdots, x_{d,n})^T, \quad d = 1, \cdots, N_p \tag{3.7}$$

A população inicial $X_d^{(0)}$ é escolhida aleatoriamente, dentro do espaço de busca, definido pelas restrições laterais, fornecidas pelo usuário:

$$x_{d,i}^{(0)} = x_i^L + r(x_i^U - x_i^L), \quad i = 1, \cdots, n \quad e \quad d = 1, \cdots, N_p \tag{3.8}$$

sendo que r é um número gerado randomicamente entre 0 e 1.

Agora com a população estabelecida, é necessário a criação de novos indivíduos, isto se dará através dos operadores que serão apresentados em seguida.

Mutação

Sejam os vetores X_α, X_β e X_γ escolhidos aleatoriamente e distintos entre si. Na geração q um par de vetores (X_β, X_γ) definem uma diferença $X_\beta - X_\gamma$. Esta diferença é multiplicada por $F > 0$, sendo denotada por diferença ponderada, e é usada para perturbar o terceiro vetor X_α ou o melhor vetor X_{best} da população. Este processo, que resulta no vetor doador $V^{(q+1)}$, pode ser escrito matematicamente como:

$$V^{(q+1)} = X_\alpha^{(q)} + F\left(X_\beta^{(q)} - X_\gamma^{(q)}\right) \quad ou \quad V^{(q+1)} = X_{best}^{(q)} + F\left(X_\beta^{(q)} - X_\gamma^{(q)}\right) \tag{3.9}$$

sendo que os índices aleatórios $\alpha, \beta, \gamma \in \{1, \cdots, N_p\}$ são inteiros distintos entre si e diferentes do índice d. O número de indivíduos da população, N_p, deve ser maior ou igual a 4. O fator de escala, ou taxa de perturbação, F, é um fator real, positivo e constante variando entre 0 e 2, cujo objetivo é controlar a amplitude da diferença ponderada. A Fig. 3.2 mostra um exemplo bidimensional que ilustra os diferentes vetores que participam da geração do vetor doador $V^{(q+1)}$.

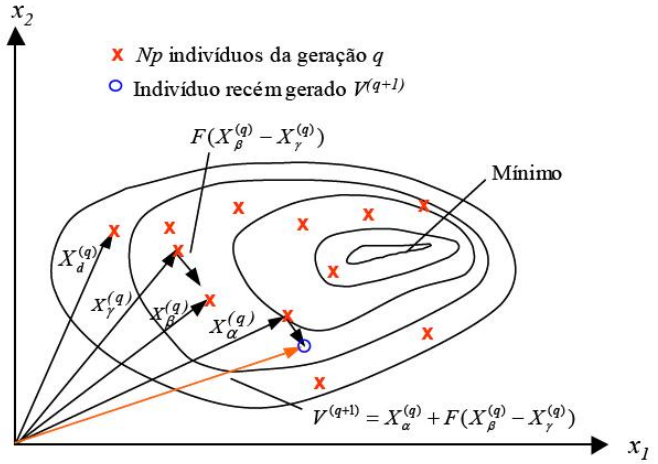


Figura 3.2 – Processo de gerar o vetor doador $V^{(q+1)}$ para uma função objetivo bidimensional. Fonte: OLIVEIRA (2011).

Se o número de indivíduos da população é grande o suficiente, a diversidade da população pode ser melhorada usando duas diferenças ponderadas para perturbar um vetor existente, ou seja, cinco vetores distintos são escolhidos aleatoriamente na população atual. O vetor diferença ponderada usa dois pares de diferenças ponderadas

e é usado para perturbar o quinto vetor ou o melhor vetor da população atual. Este processo pode ser dado por:

$$\begin{aligned} &V^{(q+1)} = X_{\alpha}^{(q)} + F\left(X_{\lambda}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right) \\ &ou \\ &V^{(q+1)} = X_{best}^{(q)} + F\left(X_{\lambda}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right) \end{aligned} \tag{3.10}$$

onde os índices aleatórios $\alpha, \beta, \delta, \gamma, \lambda \in \{1, \cdots, N_p\}$, são inteiros mutuamente distintos e diferentes do índice d, tais que $Np \geq 6$.

Existem outras maneiras, que serão apresentadas adiante, de realizar a operação de mutação, o que diferencia as diversas estratégias que podem ser utilizadas pelo método da Evolução Diferencial.

Cruzamento

Na operação cruzamento, cujo objetivo é aumentar a diversidade dos indivíduos que sofreram a mutação, as componentes do vetor doador,$V^{(q+1)}$, são misturadas com as componentes de outro indivíduo denominado vetor alvo. A escolha do vetor alvo, que deve ser diferente dos vetores já usados anteriormente, é aleatório, segundo uma probabilidade de cruzamento CR. Desta forma, obtém-se o vetor tentativa ou experimental, $U^{(q+1)}$, definido como:

$$u_i^{(q+1)} = \begin{cases} v_i^{(q+1)}, & \text{se } r_i \leq CR \\ \\ x_{d,i}^{(q)}, & \text{se } r_i > CR, \;\; i = 1, \cdots, n. \end{cases} \tag{3.11}$$

onde r_i é um número randômico entre 0 e 1 e $x_{d,i}$ são as componentes do vetor alvo $X_d^{(q)}$. A probabilidade do cruzamento ocorrer, CR, representa a probabilidade do vetor experimental herdar os valores das variáveis do vetor doador, e está compreendida entre 0 e 1, sendo fornecida pelo usuário. Quando $CR = 1$, por exemplo, todas as componentes do vetor experimental virão do vetor doador $V^{(q+1)}$. Por outro lado, se $CR = 0$, todas as

componentes do vetor experimental virão do vetor alvo $X_d^{(q)}$.

Se após o cruzamento uma ou mais componentes do vetor experimental estiver fora da região de busca, fazem-se as correções:

$$\begin{cases} \text{Se } u_i < x_i^L \text{ então } u_i = x_i^L \\ \\ \text{Se } u_i > x_i^U \text{ então } u_i = x_i^U \end{cases} \tag{3.12}$$

O cruzamento dado pela Eq. (3.11) é denominado de *cruzamento binomial* e pode ser observado na Fig. 3.3.

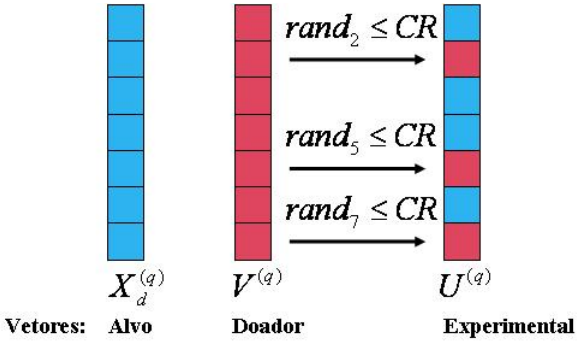


Figura 3.3 – Cruzamento binomial

Outra forma de realizar o cruzamento, denominado *cruzamento exponencial*, é definida segundo a Fig. 3.4. As componentes do vetor experimental são dadas pelas componentes do vetor doador enquanto o número randômico for menor ou igual à probabilidade de cruzamento CR.

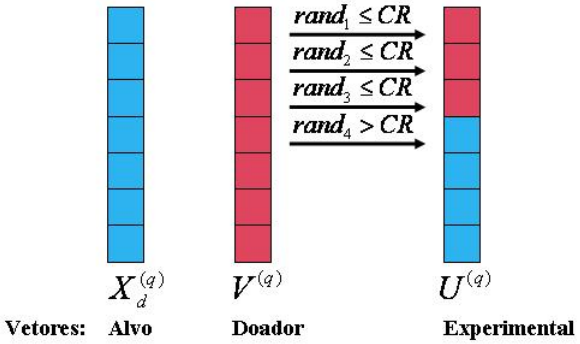


Figura 3.4 – Cruzamento exponencial

Após determinado o vetor experimental $U^{(q)}$, torna-se necessário selecionar os melhores descendentes. Esta operação será vista na próxima seção.

Seleção

A seleção é o processo de produzir melhores filhos. Diferentemente de outros algoritmos evolutivos, a evolução diferencial não usa hierarquia (elitismo) nem seleção proporcional. Em vez disso, o custo do vetor experimental $U^{(q+1)}$ é calculado e comparado com o custo do vetor alvo $X_d^{(q)}$. Se o custo do vetor alvo for menor que o custo do vetor experimental, o vetor alvo é permitido avançar para a próxima geração. Caso contrário, o vetor experimental substitui o vetor alvo na geração seguinte. Em outras palavras, este processo pode ser escrito como:

$$\begin{cases} \text{Se } f(U^{(q+1)}) \leq f(X_d^{(q)}), \text{ então } X_d^{(q+1)} = U^{(q+1)} \\ \\ \text{Se } f(U^{(q+1)}) > f(X_d^{(q)}), \text{ então } X_d^{(q+1)} = X_d^{(q)} \end{cases} \tag{3.13}$$

O processo iterativo continua até que seja alcançado algum critério de parada, sendo que um número máximo de gerações deve ser estabelecido. Para problemas com restrição, um critério pode ser a não violação das restrições ou o melhor indivíduo ter encontrado um valor dentro de uma precisão pré-estabelecida. Assim, os seguintes critérios de parada podem se estabelecidos:

$$\begin{cases} q < max \\ |f_{min}^{(q+1)} - f_{min}^{(q)}| < \epsilon \end{cases} \tag{3.14}$$

3.3.2 Parâmetros da Evolução Diferencial

O seguinte conjunto de regras pode ajudar na escolha das variáveis de controle N_p, CR e F, conforme STORN (1996):

- A população inicial deve ser gerada o mais próximo possível da superfície da função objetivo;
- Frequentemente a probabilidade de cruzamento CR deve ser considerada menor

do que um, por exemplo $CR = 0,3$. Caso não ocorra convergência, adotar $CR \in [0,8; 1]$ pode ajudar;

- Para muitas aplicações $N_p = 10D$, onde D é igual a dimensão ou ao número de variáveis, é uma boa escolha. Normalmente, F pode ser escolhido no intervalo de $[0,5; 1]$.
- Quanto maior for o tamanho da população escolhida, menor deve ser o valor de F.
- Tem-se um bom sinal de convergência quando os parâmetros do melhor componente da população variam muito de geração para geração, especialmente durante o início do processo de minimização, mesmo se seu valor da função objetivo decrescer lentamente;
- O valor da função objetivo do melhor indivíduo não pode cair de forma brusca, caso isto aconteça, a otimização está em um mínimo local;

3.3.3 Estratégias da Evolução Diferencial

Vale ressaltar que a Evolução Diferencial apresenta diferentes estratégias obtidas a partir da forma com que os operadores de mutação e cruzamento trabalham, ou seja, as estratégias da evolução diferencial podem variar de acordo com o tipo de indivíduo a ser modificado na formação do vetor doador, o número de indivíduos considerados para a perturbação e o tipo de cruzamento a ser utilizado, podendo ser escritas como: ED/a/b/c, sendo que:

a $\rightarrow$ especifica o vetor a ser perturbado, podendo ser $rand$ (um vetor da população escolhido aleatoriamente) ou $best$ (o vetor de menor custo da população);

b $\rightarrow$ determina o número de diferenças ponderadas usadas para a perturbação de a;

c $\rightarrow$ denota o tipo de cruzamento (exp: exponencial; bin: binomial).

Em 1995, Storn and Price deram o princípio de trabalho da estratégia básica usando apenas o operador cruzamento binomial (devido aos experimentos binomiais independentes), onde o cruzamento é executado em cada variável sempre que um número

$r \in [0; 1]$ aleatório for menor que a probabilidade de cruzamento CR.

Alguns anos mais tarde, Storn and Price (1997) desenvolveram mais estratégias usando o operador cruzamento exponencial, em que o cruzamento é executado nas variáveis em um laço até que esteja dentro do limite de CR. A primeira vez que um número $r \in [0; 1]$ aleatório ultrapassa o valor de CR, nenhum cruzamento é executado e as variáveis restantes são deixadas intactas. Resumidamente, as dez estratégias podem ser descritas de acordo com a Tab. 3.1.

No entanto, uma estratégia que funciona bem para um dado problema pode não funcionar bem quando aplicada a outro problema. A estratégia a ser adotada para um problema é determinada por tentativa e erro.

Tabela 3.1 – Estratégias do método Evolução Diferencial

Estratégia	Operador Mutação	Notação
1	$V^{(q+1)} = X_{\alpha}^{(q)} + F(X_{\beta}^{(q)} - X_{\gamma}^{(q)})$	ED/rand/1/bin
2	$V^{(q+1)} = X_{best}^{(q)} + F(X_{\beta}^{(q)} - X_{\gamma}^{(q)})$	ED/best/1/bin
3	$V^{(q+1)} = X_{\alpha}^{(q)} + F(X_{\lambda}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)})$	ED/rand/2/bin
4	$V^{(q+1)} = X_{best}^{(q)} + F(X_{\alpha}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)})$	ED/best/2/bin
5	$V^{(q+1)} = X_{old}^{(q)} + F(X_{best}^{(q)} - X_{old}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)})$	ED/rand-to-best/2/bin
6	$V^{(q+1)} = X_{\alpha}^{(q)} + F(X_{\beta}^{(q)} - X_{\gamma}^{(q)})$	ED/rand/1/exp
7	$V^{(q+1)} = X_{best}^{(q)} + F(X_{\beta}^{(q)} - X_{\gamma}^{(q)})$	ED/best/1/exp
8	$V^{(q+1)} = X_{\alpha}^{(q)} + F(X_{\lambda}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)})$	ED/rand/2/exp
9	$V^{(q+1)} = X_{best}^{(q)} + F(X_{\alpha}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)})$	ED/best/2/exp
10	$V^{(q+1)} = X_{old}^{(q)} + F(X_{best}^{(q)} - X_{old}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)})$	ED/rand-to-best/2/exp

Apesar de vários aspectos positivos, tem-se observado que a ED às vezes não apresenta uma performance tão boa quanto se espera. A análise empírica da ED tem mostrado que o algoritmo pode deixar de prosseguir rumo a um ótimo global tendendo a um estado de estagnação, no qual os indivíduos ficam muito parecidos entre si, isto é, a população fica homogênea e o algoritmo não apresenta qualquer melhora, apesar de aceitar novos indivíduos na população. Além disso, ED também sofre com o problema da convergência prematura. Esta situação surge quando há uma perda da diversidade da população. Como resultado, a população converge para um ponto que pode não ser uma solução ótima local. Isso geralmente ocorre quando a função objetivo é multimodal, tendo vários ótimos locais e global.

Assim como outros algoritmos evolutivos, o desempenho da ED se deteriora com o aumento da dimensionalidade da função objetivo. Várias modificações tem sido propostas na estrutura deste algoritmo a fim de melhorar seu desempenho. Na seção seguinte será tratado uma desta propostas de modificação no esquema básico da ED.

3.4 Evolução com Conjuntos Embaralhados

A idéia principal da abordagem da Evolução com Conjuntos Embaralhados (ECE) é tratar a procura global como um processo de evolução natural. Neste estudo será aplicado esse conceito para o algoritmo da ED. Os N_p pontos de amostragem, constituem uma população que é dividida em diversos conjuntos (sub-populações), tais que cada conjunto tenha o mesmo número de indivíduos. A cada um dos conjuntos é permitido evoluir de forma independente (ou seja, explorar o espaço de busca em direções diferentes). Depois de um certo número de gerações, os conjuntos são forçados a se misturar e novos conjuntos são formados através de um processo de embaralhamento. Este procedimento aumenta a capacidade de sobrevivência por um intercâmbio de informações sobre o espaço de busca adquirida de forma independente por cada conjunto.

Cada membro de um conjunto é um pai com a capacidade de participar de um processo de reprodução. Por fim, cada novo descendente substitui o seu ponto de destino correspondente no complexo atual ao invés do ponto de toda a população. Isto garante que cada pai receba pelo menos uma chance de contribuir para o processo de reprodução antes de serem substituídos ou eliminados. Assim, nenhuma das informações contidas na amostra é ignorada. O método ECE é projetado para melhorar características do método da ED, incorporando em si o poderoso conceito de conjuntos embaralhados.

Conjuntos embaralhados ajudam a garantir que as informações contidas na amostra são eficientes e completamente exploradas. Também ajudam a garantir que o conjunto de informações não se degenere. Estas vantagens do método ECE que motivam a aplicá-lo sobre o algoritmo da ED, com a esperança de se obter melhores propriedades de convergência global sobre uma ampla gama de problemas. Em outras palavras, dado um número pré-especificado de avaliações da função objetivo (nível fixo

de eficiência), espera-se que o método ECE tenha uma maior probabilidade de sucesso em seu objetivo de encontrar o ótimo global em comparação à ED.

O método ECE é baseado em uma síntese de quatro conceitos que se revelaram bem sucedidos para a otimização global: (a) combinação de abordagens aleatórias e determinísticas, (b) o conceito de computação paralela, (c) o conceito de uma evolução sistemática de um conjunto de pontos que exploram o espaço, no sentido da melhoria global, (d) o conceito de evolução competitiva.

Uma breve discussão destes conceitos é apresentada aqui. O uso de estratégias determinísticas permite que o algoritmo ECE faça uso efetivo das informações da função objetivo para orientar a pesquisa, enquanto a inclusão de elementos aleatórios ajuda a tornar o algoritmo robusto e flexível. A pesquisa começa com um conjunto de pontos selecionado ao acaso abrangendo todo o espaço de busca Ω. Um grande, porém suficiente, número de pontos ajuda a garantir que o conjunto contenha informações quanto ao número, localização e tamanho das principais regiões viáveis.

A implementação de uma estratégia de agrupamento implícito ajuda a concentrar a busca na região mais promissora identificada pelo conjunto inicial. O uso de uma estratégia de evolução sistemática ajuda a garantir que a busca seja relativamente robusta e guiada pela estrutura da função objetivo. A robustez é resultado do fato de que a estrutura de conjuntos é capaz de lidar muito bem com superfícies de função objetivo rudes, insensíveis e altamente não convexas e é relativamente pouca afetada pelos mínimos locais que são encontrados na busca pela solução global. Além disso, não são necessários informações de derivadas. A implementação de uma estratégia de evolução competitiva, conforme descrita por HOLLAND (1975), tem sido feita por ser útil na melhoria da eficiência da convergência global.

A busca aleatória controlada (*Controlled Random Search* - CRS), método descrito por Price (PRICE (1978), PRICE (1983), PRICE (1987)), foi utilizada como ponto de partida para o método ECE, uma vez que incorpora alguns dos conceitos citados acima, sendo fácil de implementar e modificar. Price propôs e testou três versões para esta estratégia (CRS1, CRS2, CRS3). Estes métodos possuem várias propriedades que são desejáveis para uma otimização global eficaz e eficiente. No entanto, eles também

têm certas fraquezas. A estratégia CRS1 trata cada região da área amostrada de forma não preferencial e a convergência pode ser lenta. Por outro lado, as estratégias CRS2 e CRS3 colocam sua ênfase principal no melhor ponto da amostra atual, de modo que a busca pode facilmente tornar-se inclinada para a região de um mínimo local. Além disso, a estratégia de sempre substituir o pior ponto atual da população por cada ponto recém-gerado faz com que a população encolha para uma pequena região muito rapidamente. Testes realizados por DUAN; GUPTA; SOROOSHIAN (1993) mostram que há uma boa possibilidade de que a população comece a conter os pontos repetidos, tornando-se degenerada e fazendo com que a busca se encerre prematuramente. Para minimizar esta possibilidade, o tamanho da amostra tem que ser suficientemente grande. No método da Evolução com Conjuntos Embaralhados, que será apresentado em seguida, este último problema não ocorre.

A estratégia Evolução com Conjuntos Embaralhados (ECE) combina os pontos fortes dos algoritmos de busca aleatória controlada com o conceito de evolução competitiva (HOLLAND (1975)), e o conceito recentemente desenvolvido de conjuntos embaralhados. O método ECE é apresentado a seguir e ilustrado na Fig. 3.5.

Passo 1: *Inicialização*. Selecione $p \geq 1$ e $m \geq n+1$, onde p é o número de conjuntos e m é o número de pontos em cada conjunto. Calcule o tamanho da amostra $s = p \times m$.

Passo 2: *Geração da amostra*. Obtenha a amostra de s pontos $x_1, \cdots, x_s$ pertencentes à região viável $\Omega \subset \mathbb{R}^n$. Calcule o valor da função f_i em cada ponto x_i. Na ausência de informação prévia, use uma distribuição uniforme de amostragem.

Passo 3: *Classificação dos pontos (rank)*. Organize os s pontos na ordem crescente do valor da função. Guarde-os em uma matriz $D = \{x_i, f_i; i = 1, \cdots, s\}$, de modo que $i = 1$ represente o ponto com o menor valor da função.

Passo 4: *Particionando em conjuntos*. Particione D em p conjuntos $A^1, \cdots, A^p$, cada um contendo m pontos, tais que:

$$A^k = \{x_j^k, f_j^k; x_j^k = x_{k+p(j-1)}, f_j^k = f_{k+p(j-1)}, j = 1, \cdots, m\}.$$

Passo 5: *Evoluir cada conjunto*. Evoluir cada conjunto $A^k, k = 1, \cdots, p$ de

acordo com o algoritmo da evolução com conjuntos competitivos (ECC) descrito separadamente.

Passo 6: *Conjuntos embaralhados*. Substitua $A^1, \cdots, A^p$ em D, tal que $D = \{A^k, k = 1, \cdots, p\}$. Ordene D em ordem de crescimento do valor da função.

Passo 7: *Testando a convergência*. Se os critérios de convergência foram satisfeitos, pare. Caso contrário, retorne ao passo 4.

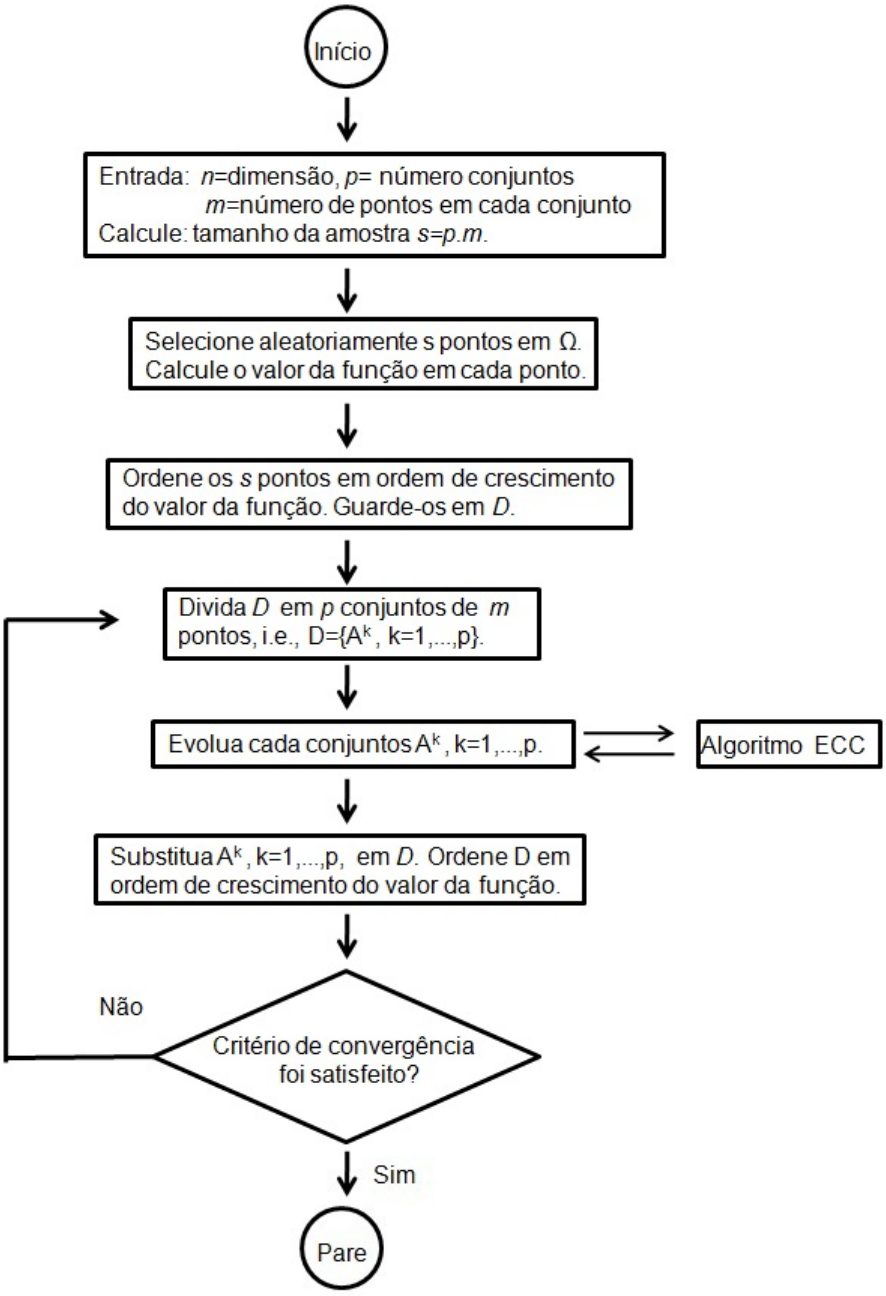


Figura 3.5 – Fluxograma do Método Evolução com Conjuntos Embaralhados - ECE.
Fonte: tradução de DUAN; GUPTA; SOROOSHIAN (1993).

A filosofia por trás da abordagem ECE é tratar a busca global como um pro-

cesso de evolução natural. Os s pontos amostrados, constituem uma população. A população é dividida em várias comunidades (conjuntos), sendo que cada uma deles é permitido evoluir de forma independente, ou seja, busca o espaço em diferentes direções. Depois de um certo número de gerações, as comunidades são forçadas a se misturar e as novas comunidades são formadas por um processo de embaralhamento. Este procedimento aumenta a capacidade de sobrevivência por um intercâmbio de informações (sobre o espaço de busca) obtido de forma independente por cada sub-população.

Cada membro de um conjunto é um pai em potencial com a capacidade de participar de um processo de reprodução. Um subconjunto selecionado a partir do conjunto é como um par de pais, salvo que um subconjunto pode ser composto por mais de dois membros. Para garantir que o processo de evolução seja competitivo, é necessário que a probabilidade de que os melhores pais contribuam para a geração de descendentes seja maior que a dos piores pais. O uso de uma distribuição de probabilidade triangular assegura esta competitividade. O procedimento desenvolvido por NELDER e MEAD (1965) é aplicado a cada subconjunto para gerar a maior parte dos descendentes. Esta estratégia utiliza as informações contidas no subconjunto para conduzir a evolução para uma direção de melhora. Além disso, os descendentes são introduzidos em locais aleatórios do espaço viável sob determinadas condições, a fim de garantir que o processo de evolução não fique preso em regiões pouco promissoras. Isso é um pouco semelhante ao processo da mutação em resposta à convergência prematura que pode ocorrer na evolução biológica. Cada mutação também ajuda a aumentar a quantidade de informações armazenadas na amostra. Por fim, cada descendente novo substitui o pior ponto do subconjunto atual, ao invés de o pior ponto da população. Isso garante que cada pai recebe pelo menos uma chance de contribuir para o processo de reprodução, antes de ser substituído ou descartado. Assim, nenhuma das informações contidas na amostra é ignorada.

3.4.1 O Método Evolução com Conjuntos Competitivos - ECC

O algoritmo da Evolução com Conjuntos Competitivos (ECC) necessário para a atualização de cada conjunto no Passo 5 do método ECE é apresentado a seguir e é

ilustrado na Fig. 3.6.

Passo 1: *Inicialização*. Selecione q, α, e β, onde $2 \leq q \leq m, \alpha \geq 1$, e $\beta \geq 1$.

Passo 2: *Atribuir pesos*. Atribuir uma distribuição de probabilidade triangular para A^k, isto é,

$$p_i = 2(m+1-i)/m(m+1), \qquad i = 1, \cdots, m. \tag{3.15}$$

Observando a Eq. (3.15), verifica-se que o ponto x_1^k ($i = 1$) tem a maior probabilidade, $p_1 = 2/(m+1)$. O ponto x_m^k ($i = m$) tem a menor probabilidade, $p_m = 2/m(m+1)$.

Passo 3: *Seleção dos pais*. Escolher aleatoriamente q pontos distintos $u_1, \cdots, u_q$ de A^k de acordo com a distribuição de probabilidade acima especificada (os q pontos definem um sub-conjunto). Armazene-os na matriz $B = \{u_i, v_i, i = 1, \cdots, q\}$ onde v_j é o valor da função associado ao ponto u_j. Armazene em L as localizações de A^k que são usadas para construir B.

Passo 4: *Geração de descendentes*.

(a) Ordenar B e L de modo que os q pontos estejam dispostos em ordem crescente do valor da função. Calcule o centróide g, utilizando a expressão:

$$g = [1/(q-1)] \sum_{j=1}^{q-1} u_j. \tag{3.16}$$

(b) Calcule o novo ponto (etapa de reflexão)

$$r = 2g - u_q. \tag{3.17}$$

(c) Se $r \in \Omega$, calcule o valor da função f e vá para a etapa (d), caso contrário, calcule o menor hipercubo $H \subset \mathbb{R}^n$ que contenha A^k, gere aleatoriamente um ponto $z \in H$, calcule f_z, defina $r = z$ e $f_r = f_z$ (etapa mutação).

(d) Se $f_r < f_q$, substitua u_q por r e vá para a etapa (f); caso contrário, (etapa contração) calcule

$$c = (q + u_q)/2 \quad e \quad f_c. \tag{3.18}$$

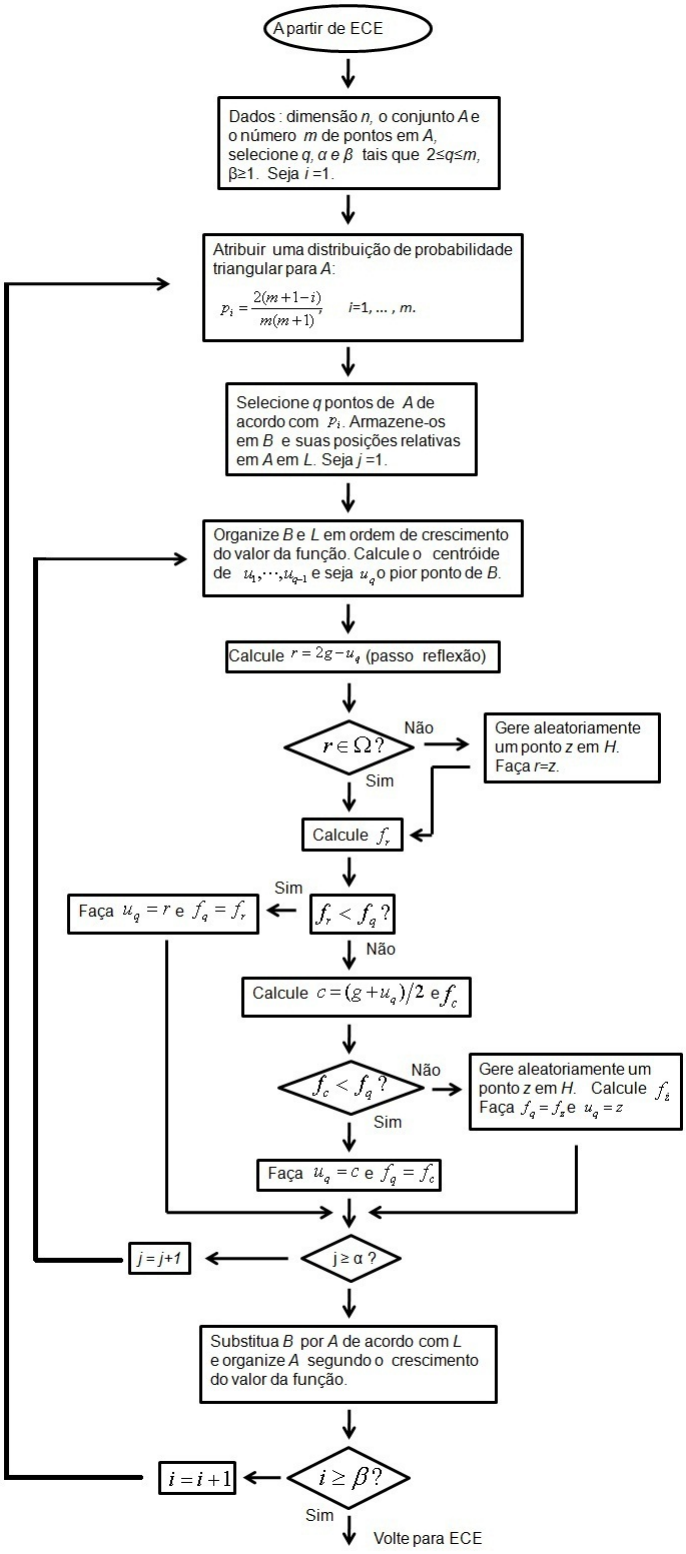


Figura 3.6 – Fluxograma do Método Evolução com Conjuntos Competitivos - ECC.
Fonte: tradução de DUAN; GUPTA; SOROOSHIAN (1993).

(e) Se $f_c < f_q$, substitua u_q por c e vá para a etapa (f); caso contrário, gere aleatoriamente um ponto $z \in H$ e calcule f_z (etapa mutação). Substitua u_q por z.

(f) Repita α vezes as etapas de (a) até (e), sendo que $\alpha \geq 1$ é um parâmetro especificado pelo usuário.

Passo 5: *Substituição dos pais pelos descendentes*. Substitua B por A^k usando a localização original armazenada em L. Ordene A^k segundo o crescimento do valor da função.

Passo 6: *Iteração*. Repita β vezes os passos de (1) até (4), onde $\beta \geq 1$ é um parâmetro especificado pelo usuário que determina quantos filhos deverão ser gerados (o quanto cada conjunto deverá evoluir).

3.5 Evolução Diferencial Melhorada - EDM

O algoritmo da Evolução Diferencial Melhorada (EDM) proposto é uma combinação da evolução com conjuntos embaralhados e a evolução diferencial básica (ED).

A EDM inicia-se como o algoritmo da ED usual por criar uma população de indivíduos amostrados aleatoriamente a partir da região viável usando distribuição de probabilidade uniforme. A população é então classificada em ordem crescente de valores da função objetivo e particionada em diversos conjuntos. Cada conjunto independentemente executa a ED. Na etapa da evolução, os conjuntos são obrigados a se misturar e os pontos são realocados para garantir a troca de informações. O processo do algoritmo EDM proposto é descrito a seguir e os seu fluxograma é apresentado na Fig. 3.7.

Passo 1: *Inicialização*. Gerar aleatoriamente uma população inicial de N_p vetores $X_{i,j}$, sendo que cada vetor tem dimensão n, usando a seguinte regra:

$$X_{i,j} = X_{min,j} + rand(X_{max,j} - X_{min,j}), \tag{3.19}$$

onde $X_{min,j}$ e $X_{max,j}$ são os limites inferior e superior para a j-ésima componente respectivamente e $rand$ um número aleatório uniforme entre 0 e 1. Calcule o valor da função objetivo $f_i = f(X_i)$ para todo X_i. Defina o número máximo de geração como G_{max}. Seja

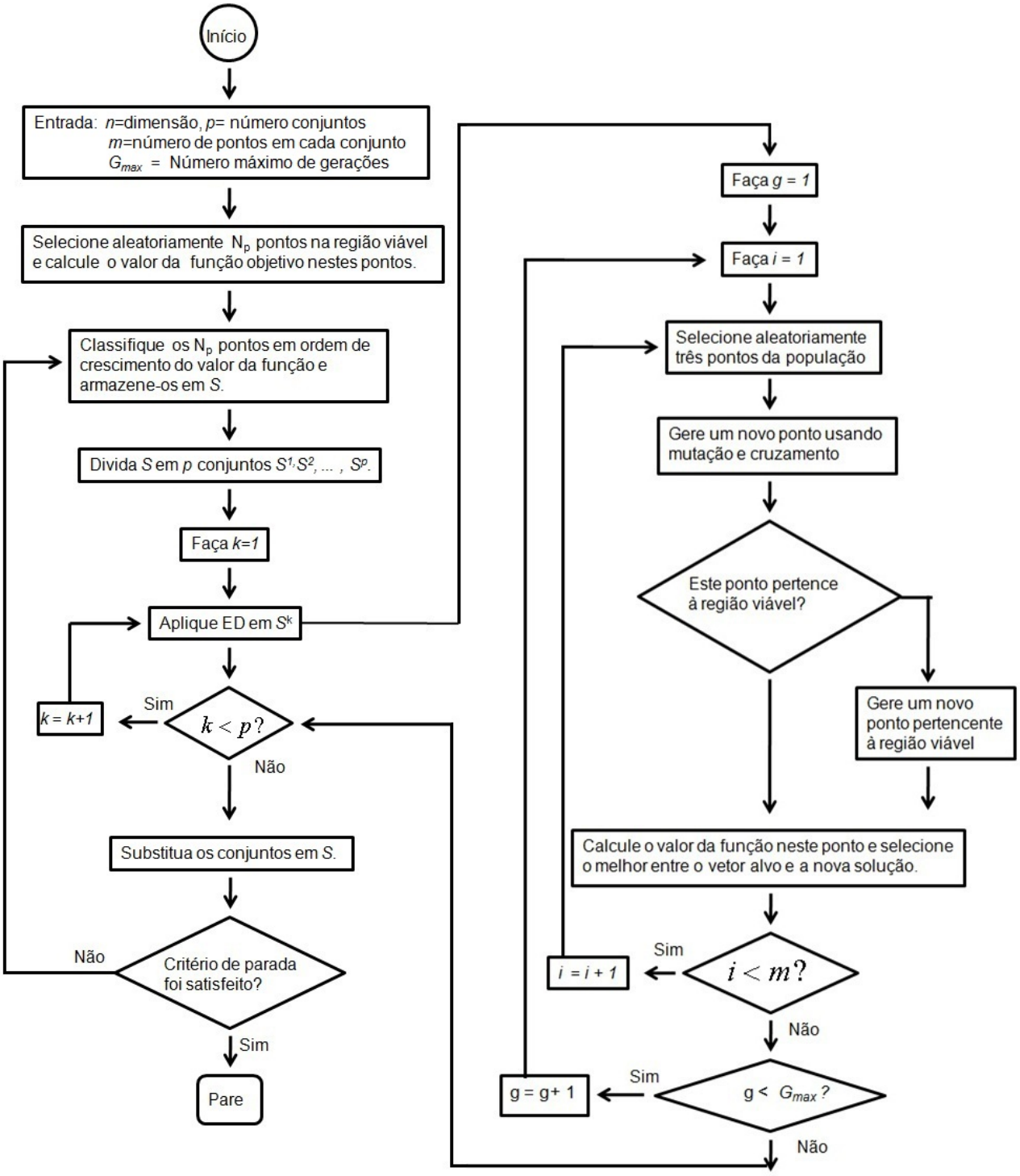


Figura 3.7 – Fluxograma do Método Evolução Diferencial Melhorada - EDM.

$N_p = p \times m$, onde p é o número de conjuntos e m é o número de indivíduos em cada conjunto.

Passo 2: *Classificação*. Classifique a população inteira em ordem de crescimento do valor da função objetivo. Armazene-os em um conjunto $S = \{X_i, f_i : i = 1, \cdots, N_p\}$.

Passo 3: *Divisão*. Divida S em p subpopulações $S^1, S^2, \cdots, S^p$, cada uma contendo m pontos, tais que:

$$S^k = \{X_j^k, f_j^k : X_j^k = X_{k+p(j-1)}, f_j^k = f_{k+p(j-1)}, j = 1, \cdots, m\}, \qquad k = 1, 2, \cdots, p.$$

Passo 4: *Evolução*. Seja $k = 1$.

Passo 4.1: Aplique ED a cada subpopulação S^k.

Passo 4.2: *Inicialização*. Defina o contador da geração $g = 1$. O conjunto S^k trabalha como uma população em ED.

Passo 4.3: *Mutação*. Selecione aleatoriamente três pontos da população S^k e gere o vetor doador V_i utilizando a Eq. (3.9).

Passo 4.4: *Cruzamento*. Recombine cada vetor alvo X_i com o vetor doador gerado no passo 4.3 para gerar o vetor experimental U_i utilizando a equação (3.11).

Passo 4.5: *Verificação de viabilidade*. Verifique se cada variável do vetor experimental pertence à região viável. No caso afirmativo, vá para o passo 4.6, caso contrário, corrija-o da seguinte forma:

$$U_{i,j} = \begin{cases} 2X_{min,j} - U_{i,j} & \text{se } U_{i,j} < X_{min,j} \\ \\ 2X_{max,j} - U_{i,j} & \text{se } U_{i,j} > X_{max,j} \end{cases} \tag{3.20}$$

e vá para o passo 4.6.

Passo 4.6: *Seleção*. Calcule o valor da função objetivo para o vetor U_i. Escolha o melhor vetor comparando o valor da função objetivo dos vetores alvo e experimental usando a Eq. (3.13) para a próxima geração.

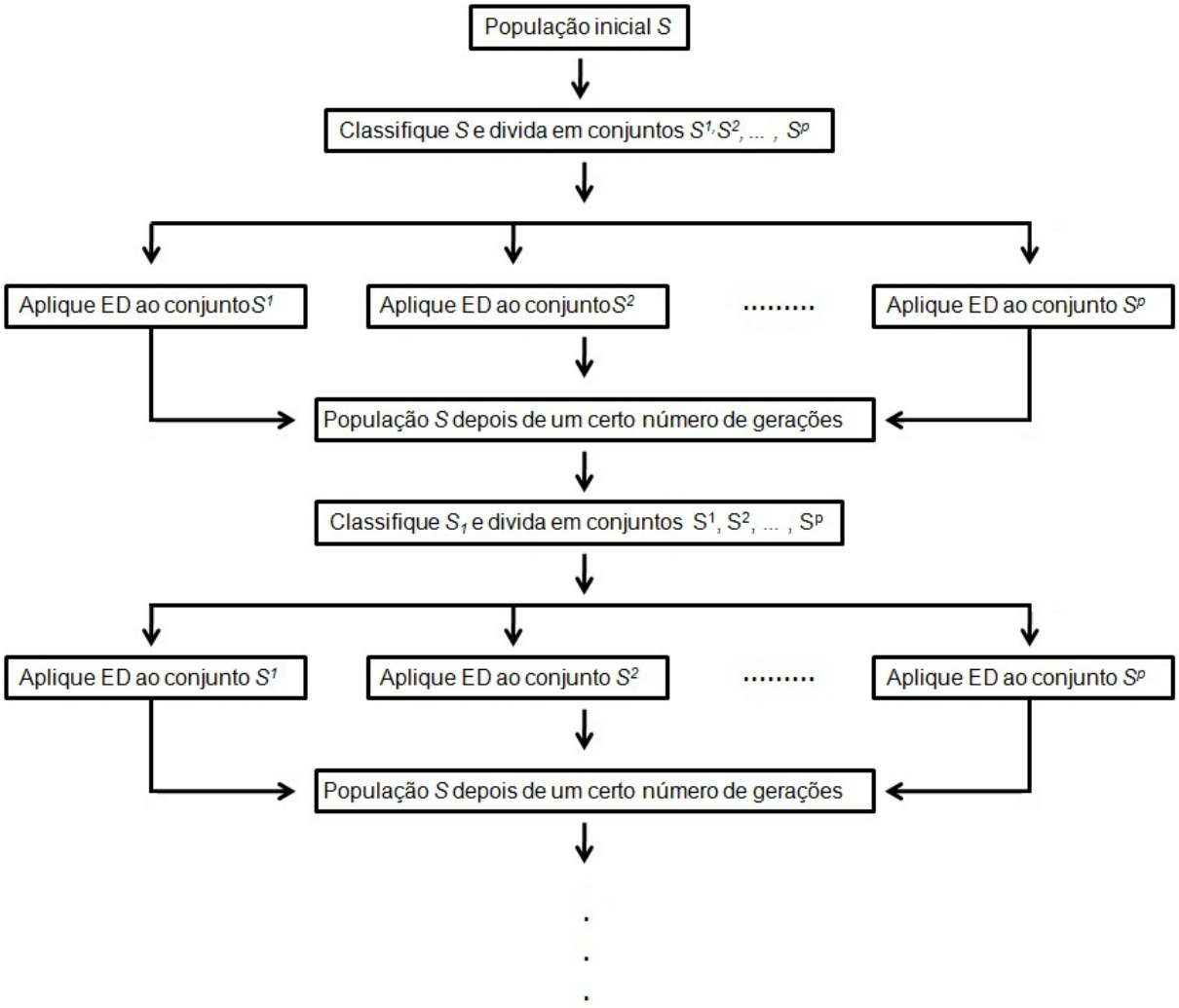


Figura 3.8 – Arquitetura do Método Evolução Diferencial Melhorada - EDM.

Passo 4.7: *Iteração*. Se $g < G_{max}$ então vá para o passo 4.3 com $g = g + 1$, caso contrário, vá para o passo 5.

Passo 5: Se $k < p$ então $k = k + 1$ e vá para o passo 4.1, caso contrário, vá para o passo 6.

Passo 6: *Embaralhando os conjuntos*. Substitua os conjuntos $S^1, S^2, \cdots, S^p$ em S e verifique se os critérios de parada foram satisfeitos, se sim, pare, caso contrário vá para o passo 2.

A partir do algoritmo apresentado várias outras versões já foram propostas alterando o número de conjuntos (subpopulações) utilizados e também o número de vezes que o algoritmo da ED básica é chamado para cada conjunto antes do processo de embaralhamento. A arquitetura do método EDM é ilustrado na Fig. 3.8.

3.5.1 Evolução Diferencial Melhorada Utilizando Processamento Paralelo

Em uma sociedade capitalista envolta de curtos prazos e visando sempre o lucro, quanto mais rápido o algoritmo conseguir resolver um problema melhor. A ideia de se dividir tarefas de programas por vários processadores é antiga, mas só recentemente vem se tornando viável devido ao rápido avanço de *hardware* e *software*. Segundo o grupo de pesquisa em Processamento da Linguagem Natural da Pontifícia Universidade Católica do Rio Grande do Sul (PLN (2014)), as máquinas evoluiram muito em velocidade e com o aparecimento de máquinas com vários processadores a velocidade de comunicação vem apresentando uma grande evolução permitindo que estes processadores troquem informações com rapidez; e os programas que possibilitam esta comunicação vêm mostrando grande aumento de eficiência. Dessa forma, o objetivo é diminuir o tempo de processamento durante a execução do algoritmo melhorado, para permitir sua aplicação em problemas complexos.

Depois de combinar as potencialidades dos métodos da Evolução Diferencial e da Evolução com Conjuntos Embaralhados é preciso implementá-los em paralelo, pois assim, cada conjunto do método ECE evoluirá em um processador diferente e ao mesmo tempo, seguindo o princípio de que grandes problemas geralmente podem ser divididos em problemas menores, para então serem resolvidos concorrentemente (em paralelo).

No processador mestre é criado randomicamente a população inicial, a qual é divida em k subpopulações ($k \in \mathbb{Z}$ é menor ou igual ao número de processadores) e distribuída pelos no máximo k processadores. Em seguida, o código da ED segue processando sequencialmente em cada processador até atingir algum critério de parada. As subpopulações são então reagrupadas no processador mestre, embaralhadas e divididas novamente em subpopulações.

Resumidamente, o Método de Otimização Evolução Diferencial Melhorada com Processamento Paralelo (EDMP) une o que há de melhor nos métodos ED e ECE em um só algoritmo que é implementando em paralelo. O fluxograma do Método de Otimização Evolução Diferencial Melhorada em Paralelo pode ser visto na Fig. 3.9.

O algoritmo da EDMP foi implementado em C++ e suas primeiras simulações

foram realizadas em um Cluster Beowulf de 36 processadores.

Inicialmente, o algoritmo da Evolução Diferencial utilizava apenas as operações: mutação, cruzamento e seleção para evoluir a população. Mas, após algumas simulações numéricas, percebeu-se que o algoritmo da ED algumas vezes não alcançava resultados tão bons quanto ao ECE, e isto muitas vezes acontecia por que a população obtida após algumas iterações utilizando ED se tornava homogênea e o código computacional encerrava prematuramente a busca pela solução ótima. Para resolver este problema, foi utilizado a ideia do algoritmo ECE de embaralhar a população em cada nova iteração antes que esta evolua por meio das três operações da ED. Desta forma começou surgir o algoritmo EDM.

Mas, por meio da realização de novos testes, observou-se que o algoritmo da EDM ainda não estava tão rápido, no sentido de tempo computacional, quanto era preciso. Por isso, tornou-se necessário pensar numa estratégia para acelerar o algoritmo, foi então que surgiu a ideia de elaborar o algorimo da EDM em paralelo, pois desta forma poder-se-ía dividir as tarefas pelos processadores disponíveis no ambiente de trabalho do usuário.

Deu-se então início aos estudos da programação paralela bem como do uso da biblioteca Message Passing Interface (MPI). De início, pretendia-se elaborar a EDM em paralelo no software Matlab® pois tanto o código computacional da ED como da ECE foram escritos para a linguagem do Matlab®. Assim, foi adquirido uma licença do Matlab® para Linux bem como o Parallel Computing Toolbox, pois este toolbox permite resolver problemas computacionais com grande volume de dados usando processadores multicore, as Graphics Processing Unit (GPUs) ou Unidade de Processamento Gráfico, e os clusters de computadores. Mas surgiram alguns imprevistos: para que o Parallel Computing Toolbox funcionasse em um cluster de computadores era necessário adquirir o MATLAB Distributed Computing Server e em cada computador deveria ter o Matlab® instalado. Esta instalação representaria altos custos e o algoritmo ficaria “preso” às máquinas que tivessem o Matlab® instalado. Desta forma, optou-se por escrever o código em C++ e processá-lo em máquinas com Linux.

O primeiro e mais importante passo para escrever o código da EDM em paralelo

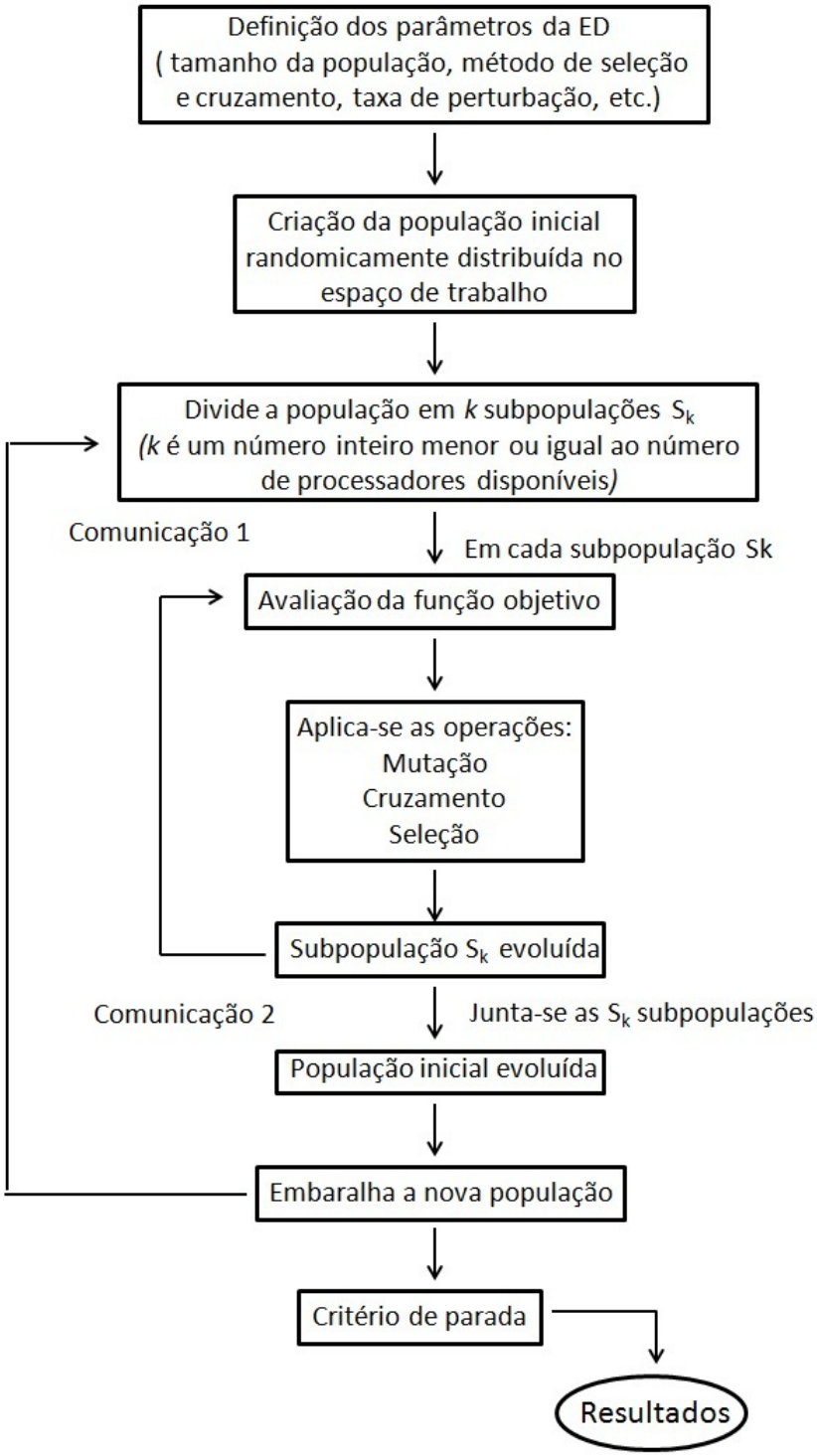


Figura 3.9 – Algoritmo Evolução Diferencial Melhorada com Processamento Paralelo.

foi decidir em que momentos haveria comunicação entre os processadores pois quanto maior for a comunicação menor é a eficiência do algoritmo em paralelo. Novamente, foi utilizado a ideia da ECE de dividir a população inicial em complexos ou subpopulações. Em seguida, cada subpopulação iria evoluir separadamente usando a ED até que algum critério de parada referente a ED fosse satisfeito. A seguir, as subpopulações serão reagrupadas, embaralhadas e dividas novamente em subpopulações e o processo continua até satisfazer algum critério de parada da ECE. No processador mestre é criado randomicamente a população inicial, a qual é dividida em k subpopulações ($k \in \mathbb{Z}$ é menor ou igual ao número de processadores) e distribuída pelos no máximo k processadores, a primeira comunicação entre os processadores acontece aqui. Em seguida, o código da ED segue processando sequencialmente em cada processador até atingir algum critério de parada. As subpopulações são então reagrupadas no processador mestre, que é a segunda comunicação entre os processadores, embaralhada e dividida novamente em subpopulações. Essas duas comunicações entre os processadores são feitas por meio do comando MPI_allgatherv, cuja função é reunir os dados e distribuir os dados combinados de todas as tarefas.

Com o código “pronto” iniciou-se algumas verificações utilizando funções testes a fim de se encontrar possíveis erros de implementação bem como verificar a capacidade de otimização do código. As funções testes de otimização utilizadas possuem características distintas: restritas, irrestritas, multimodais e uni-modais. Todos os problemas testes usados para validação do código foram consultados no site OPTIMA (2012) e suas simulações serão apresentadas no Capítulo V.

Depois de vários testes percebeu-se que para alguns problemas o algoritmo não conseguia “escapar” dos mínimos locais. Ao listar os membros da população notou-se que esta tornava-se muito homogênea, as subpopulações provenientes de processadores diferentes algumas vezes eram praticamente iguais. Para evitar isso, em cada nova iteração da EDM ao invés de juntar todas as subpopulações em uma só, 50% dos membros da nova população proviam das subpopulações evoluídas e os outros 50% seriam gerados aleatoriamente dentro da região viável do problema. Desta forma, conseguiu-se evitar a parada prematura do algoritmo por aumentar a diversidade da população.

Novamente, foram utilizadas as funções testes a fim de validar o código, que após várias análises não apresentou problemas de implementação nem de estagnação, confirmando assim sua eficácia.

CAPÍTULO IV

NOÇÕES SOBRE PROCESSAMENTO PARALELO

O processador, seja ele em computadores, note/netbooks, tablets ou GPS's, é uma das principais ferramentas para se solucionar problemas no dia a dia, principalmente quando estes problemas estão relacionados à criação ou modelagem de sistemas mecânicos ou físicos encontrados em várias aplicações. A facilidade para se implementar algoritmos matemáticos, utilizando linguagem de programação de alto nível, tem levado a comunidade científica a estudar problemas cada vez mais complexos.

Um cluster é um sistema que atende às aplicações que podem ser divididas em tarefas, que poderão, assim, ser executadas simultaneamente por um certo número de processadores. Segundo a Agência Brasileira de Notícias - ABN (2013), praticamente todos os setores estão adotando clusters Linux a fim de obter as melhorias de desempenho necessárias para cumprir metas organizacionais. Ainda segundo a ABN, a computação de alto desempenho é aproveitada em áreas como análise sísmica para exploração de petróleo, simulação aerodinâmica para projetos de motores e aeronaves, efeitos especiais de Hollywood, modelagem molecular para pesquisa biomédica, computação empresarial super-escalável e mineração de dados e modelagem financeira para análise de negócios.

Sabe-se que não existe um algoritmo universal que resolva qualquer tipo de problema. Além disso, alguns algoritmos demandam um tempo de processamento ele-

vado e muitos recursos computacionais para encontrar algum resultado. Algumas vezes não se conhece formas para melhorar a performance do algoritmo que está sendo aplicado, outras vezes o algoritmo implementado é um código comercial, o que impossibilita modificações estruturais no algoritmo. Nestes casos, pode-se recorrer a outros recursos para melhorar o desempenho do código. Uma forma de fazer isto é por meio da computação paralela.

A busca por maior eficiência computacional no processamento em computação de alto desempenho (High Performance Computing - HPC), medida em termos do tempo necessário a obter-se a solução de um problema, tem estimulado a procura por novas estratégias para se alcançar elevadas capacidades de processamento. Atualmente, há, pelo menos, duas alternativas para a HPC, uma delas é a utilização dos super computadores, que são unidades de processamento de grande desempenho e a outra são vários computadores em paralelos. Porém, o custo de implantação e manutenção de super computadores é altíssima, por este motivo a utilização de computadores pessoais (PCs) para formar unidades de processamento em paralelo tem sido, hoje, a alternativa mais atraente.

Um exemplo de um super computador é encontrado no site TOP500 (2014). Neste site há um lista dos supercomputadores mais poderosos do mundo e em quarto lugar da lista está o “Computador K”, instalado no RIKEN Instituto Avançado de Ciência Computacional em Kobe, no Japão, que alcançou em novembro de 2011 um impressionante 10,51 Petaflop/s, ou seja, 10 quatrilhões de cálculos por segundo, usando 705.024 núcleos de processamento SPARC64.

Como o objetivo de um algoritmo paralelo é ter um tempo de execução inferior à versão sequencial é útil que se crie primeiro a versão sequencial do algoritmo do problema que se pretende resolver para então adaptá-lo à versão paralela ou, se possível, que se use o algoritmo serial como ponto de partida, pois desta forma, além de ajudar a compreender o problema o algoritmo serial também pode auxiliar na validação do algoritmo em paralelo.

Para se obter algoritmos paralelos eficientes é preciso considerar, pelo menos, cinco fatores importantes, a saber: o balanceamento de carga, isto é, dividir de forma

igual o trabalho entre os processadores; minimizar as necessidades de comunicação; minimizar o tempo ocioso; sobrepor as operações de comunicação e de computação e finalmente, mas não menos importante, minimizar as operações de entrada e saída (E/S). Estas questões serão consideradas mais adiante.

Também é necessário decidir se o paralelismo será com memória distribuída ou compartilhada. No primeiro caso, a comunicação é feita por meio do acesso à memória principal e por isso limita-se a um número relativamente pequeno de processadores. No paralelismo com memória distribuída cada processador tem acesso somente à memória a ele alocada e o acesso aos dados da memória de outros processadores ocorre via troca de mensagens entre os processadores por meio de uma rede de conexão. Neste trabalho será utilizado paralelismo com memória distribuída.

Visto que há um alto custo em se utilizar multiprocessadores os algoritmos apresentados aqui serão implementados em clusters de computadores pessoais que são economicamente mais viáveis e com poder de processamento próximo ao das máquinas paralelas.

4.1 Cluster

Segundo BUENO (2002) o termo “cluster de computadores refere-se a utilização de diversos computadores (heterogêneos ou não) conectados em rede para realização de processamento paralelo. Ou seja, as máquinas são conectadas via rede para formar um único computador”. Desta forma, o cluster dá aos usuários a impressão de um único sistema funcionando quando na verdade podem haver dezenas, centenas ou até milhares de computadores. Dessa forma, é possível realizar processamentos que até então somente computadores de alta performance seriam capazes de fazer.

Cada computador de um cluster é denominado nó, há um nó sevidor e os outros são chamados de nós clientes, os quais são interconectados via rede que pode ser Ethernet ou outra topologia qualquer. É interessante que esta rede seja criada de forma a permitir o acréscimo ou a retirada de um nó sem, no entanto, interromper o funcionamento do cluster, pois se houver danos a um ou mais computadores, por exemplo, é

possível realizar os reparos sem prejudicar o processamento. Além disso, o sistema operacional usado nos computadores deve ser o mesmo, isto é, ou somente Windows, ou somente Linux, por exemplo, visto que há particularidades em cada sistema operacional que poderiam impedir o funcionamento do cluster.

Depois de escolhido o sistema operacional, é preciso usar um software que permita a implementação de estruturas de Máquinas Virtuais Paralelas (PVM - Parallel Virtual Machines) e/ou Interface de Passagem de Mensagens (MPI - Message Passing Interface). Esse software será responsável por controlar as trocas de mensagens do cluster, distribuir os arquivos e as tarefas para os nós clientes e servir de porta de entrada e saída para conexão com uma rede externa, quando for o caso.

É claro que, quanto mais computadores existirem no cluster, maiores serão os custos de implementação e manutenção. Este aumento não se deve apenas ao preço dos computadores, mas também pelos equipamentos necessários para a construção de um cluster, como por exemplo switches, cabos, hubs, nobreaks, etc. Mesmo assim, os custos costumam ser menores do que a aquisição e/ou manutenção de supercomputadores, sendo que, em alguns casos, o processamento costuma ser até mais rápido.

Os clusters tem se mostrado bastante úteis, principalmente para aplicações que exijam processamento pesado como as de previsão do tempo e de abalos sísmicos, ou para aplicações que não podem parar de funcionar ou quando não podem ocorrer perda de dados como a dos sistemas de bancos.

O princípio de funcionamento de um cluster é simples: o servidor divide as tarefas e as distribui através do switch para os clientes. Em seguida, cada cliente recebe as mensagens e um conjunto de dados a serem processados, quando estes processamentos estiverem concluídos, os resultados são enviados novamente para o servidor, que os agrupam e finaliza o processamento.

A troca de mensagens entre servidor e clientes é realizada por meio de uma biblioteca de comunicação, as mais usuais são a PVM e a MPI. A escolha da biblioteca é determinada pelo tipo de dado a ser transmitido, confiabilidade, desempenho e número de clientes. O uso de bibliotecas de comunicação é importante pois visa minimizar os problemas decorrentes da necessidade de sincronização, uma vez que as tarefas exe-

cutadas em paralelo aguardam a finalização mútua para poderem então coordenar os resultados ou trocarem dados e reiniciar novas tarefas em paralelo.

Segundo BUENO (2002), existem diferentes tipos de estruturas utilizadas para implementar o processamento paralelo bem como diferentes clusters. O processamento paralelo pode ser implementado utilizando: (1) Swar (Simd Within a Register), o qual consiste em utilizar as instruções MMX (MultiMedia eXtensions) disponibilizadas nos processadores para realizar tarefas em paralelo, e neste caso o processamento paralelo pode ser realizado em uma máquina com um único processador; (2) com SMP (Symetric Multi Processor), isto é, por meio de computadores com mais de um processador com as mesmas características, daí vem sua denominação; (3) com cluster de uma estação de trabalho que é um conjunto de computadores completos (com teclado, monitor, mouse), conectados em rede, e que cumprem duas funções; o uso diário, com diversos tipos de programas como processadores de texto e planilhas, e o uso para processamento paralelo pesado no final do dia e/ou nos fins de semana; (4) cluster para balanceamento de carga, no qual a distribuição de processamento aos nós do cluster é equilibrada; (5) em um cluster com MOSIX, o que trata de uma extensão para Linux (ou sistemas baseados em Unix) de um sistema de cluster que trabalha como se fosse um único supercomputador, por meio de conceitos de Distribuição de Processos e Balanceamento de Carga; (6) com cluster Beowulf, sendo uma tecnologia de cluster que agrupa computadores trabalhando no sistema GNU/Linux para formar um supercomputador virtual via processamento paralelo (distribuído).

Será utilizado neste trabalho este último tipo de estrutura para implementar o processamento paralelo, isto é, o cluster Beowulf. A seguir, esta tecnologia será apresentada de maneira mais detalhada.

4.2 Clusters Beowulf

Os Clusters Beowulf são construídos a partir de sistemas de computadores pessoais (PC) disponíveis no mercado, por este motivo tem se tornado uma escolha alternativa para a construção de sistemas de computação paralela de alta performance.

O rápido avanço dos microprocessadores, das redes de interconexões de alta velocidade e das tecnologias de outros componentes têm facilitado muitas implementações bem sucedidas neste tipo de cluster.

Em seguida, será apresentado algumas definições e curiosidades sobre clusters Beowulf. Partes do texto desta seção 4.2 é uma tradução de HSIEH (2000).

O conceito de clusters Beowulf originou no Center of Excellence in Space Data and Information Sciences (CESDIS), localizado na NASA Goddard Space Flight Center, em Maryland. O primeiro cluster Beowulf foi desenvolvido em 1994 por Thomas Sterling e Don Becker, pesquisadores do CESDIS, para utilizá-lo em um projeto de Ciências Espaciais e Terrestres (ESS) em conjunto com o grupo de Computação e Comunicação de Alta Performance (HPCC) do qual os pesquisadores faziam parte. O protótipo inicial era um cluster de 16 processadores DX4 ligados por dois canais Ethernet acoplados (Ethernet bonding). A máquina foi um sucesso instantâneo e esta ideia rapidamente se espalhou pelos meios acadêmicos, pela NASA e por outras comunidades de pesquisa.

O objetivo da construção de um cluster Beowulf é de criar um sistema de computação paralela com custo-benefício para atender à sociedade de consumo de massa visto que os componentes podem ser facilmente encontrados à venda, satisfazendo, assim, os requisitos computacionais específicos da comunidade científica.

Beowulf é o nome de um herói escandinavo mitológico, conhecido através de um poema anglo-saxão medieval. Este herói é descrito no poema com uma força descomunal e como um homem de grande coragem e fortaleza, que lhe permitiu realizar grandes façanhas tanto na guerra como na batalha contra seres fantásticos. Neste conto épico, Beowulf salva o reino dinamarquês de Hrothgar de dois monstros antropófagos (Grendel e sua mãe) e de um dragão, matando cada um deles, mas termina morrendo pela gravidade das feridas da luta com tal dragão.

Beowulf é utilizado atualmente como uma metáfora para uma nova estratégia de computação de alta performance, que explora tecnologias do mercado de consumo popular para economizar tempo e dinheiro impostos pela supercomputação, liberando as pessoas para se dedicarem às suas próprias áreas de conhecimento. Ironicamente, a construção de um cluster Beowulf é tão divertido que os cientistas e pesquisadores an-

siosamente arregaçaram as mangas para realizar as muitas tarefas tediosas envolvidas, pelo menos, na primeira vez que contruíram um.

4.2.1 Definições

Visto que entre os progamadores é usado comumente alguns termos técnicos em inglês, nesta seção será dada uma breve explicação para tais termos a fim de tornar o estudo em programação paralela mais dinâmico. Todos os significados desta subseção 4.2.1 foram retirados de MORINOTO (2013).

Gigaflops: Medida de desempenho, bilhões de operações de ponto flutuante que um processador pode executar por segundo.

Gateway: Pode ser traduzido como “portão de entrada”. O gateway pode ser um PC com duas (ou mais) placas de rede, ou um dispositivo dedicado, utilizado para unir duas redes. Existem vários usos possíveis, desde interligar duas redes que utilizam protocolos diferentes, até compartilhar a conexão com a Internet entre várias estações. O endereço do gateway deve ser informado nas propriedades de rede, mas numa rede onde as estações estão configuradas para obter seus endereços automaticamente é possível configurar o servidor DHCP para enviar o endereço do gateway automaticamente.

A estação enviará ao gateway qualquer requisição de endereço que não faça parte da rede local. Se, por exemplo você tiver uma rede com 3 micros, configurados com os endereços 192.168.0.1, 192.168.0.2 e 192.168.0.3, qualquer endereço fora do escopo 192.168.0.x será enviado ao gateway, que se encarregará de acessá-lo na outra rede, ou na Internet e entregar o resultado à estação.

Ethernet É o padrão de rede mais usado atualmente. O padrão consiste em placas de rede, cabos, hubs e outros periféricos de rede compatíveis entre sí. Existem basicamente dois padrões Ethernet: 10 e 100 megabits, que se diferenciam pela velocidade. Uma placa Ethernet 10/10 transmite dados a 10 Mbits, enquanto uma 10/100 transmite a 100 Mbits, podendo transmitir também a 10 caso esteja ligada a uma placa 10/10.

Existe ainda um padrão relativamente novo, o Gigabit Ethernet, 10 vezes mais rápido que o anterior, já está em uso, apesar do alto preço. O próximo padrão, de 10

Gigabits, já está estabelecido, apesar dos protótipos disponíveis terem ainda um longo caminho de aperfeiçoamento e redução de custo até tornarem-se o novo padrão da indústria.

Middleware Um programa que permite que dois sistemas diferentes possam se comunicar. Por exemplo, permitir que um determinado programa, capaz de acessar um determinado banco de dados, possa acessar bancos de dados em outros formatos. Outro exemplo, é a possibilidade de permitir que servidores de diferentes plataformas trabalhem em conjunto. Uma das possibilidades é trabalhar com servidores Alpha e Linux combinados num sistema de processamento paralelo.

Network File System - NFS Este é o protocolo de compartilhamento de arquivos nativo do Linux e de outras versões do Unix. O NFS roda sobre o TCP/IP e foi originalmente desenvolvido pela Sun, mas é aberto e justamente por isso muito usado. A funcionalidade é parecida com a do compartilhamento de arquivos do Windows e a configuração é bastante simples. Mas, infelizmente o Windows não oferece suporte ao NFS, apesar de alguns códigos comerciais adicionarem o recurso. Apesar disso o NFS é a melhor opção para compartilhar arquivos entre máquinas trabalhando com o sistema Linux.

Telnet permite acesso remoto à qualquer máquina que esteja processando o módulo servidor (assim como no SSH) mas é inseguro, pois os dados não são criptografados. Manter o servidor Telnet ativo representa um grande risco numa máquina conectada à Internet, pois qualquer um que descubra uma das senhas de usuário, ou pior, a senha de root, terá acesso à sua máquina, o que não é desejável.

O comando existe tanto no Linux, quanto no Windows (no prompt do MS-DOS). O serviço Telnet permite acesso via terminal, pode até mesmo abrir aplicativos de modo texto, como o Links, Vi, EMACs, etc. além de permitir o uso de todos os comandos.

Commodity Trata-se um termo inglês que significa mercadoria e é utilizado para se referir aos produtos de origem primária, ou seja, em estado bruto ou com baixo grau de industrialização. São produzidos em grandes volumes por uma diversidade de produtores, além disso, podem ser estocados por um determinado período sem perda de qualidade. Café, trigo, água, borracha, ouro e petróleo são alguns exemplos de commo-

dities. Há algumas definições que descrevem o produto commodity como padronizados e de baixo valor agregado.

4.2.2 Clusters de um Sistemas de Computação Commodity

Após o sucesso do primeiro cluster Beowulf, vários outros foram construídos pelo CESDIS usando várias gerações e famílias de processadores e interconexões de rede. Durante o Supercomputing '96, uma conferência de supercomputação patrocinada pela Association for Computing Machinery (ACM) e pelo Institute of Electrical and Electronics Engineers (IEEE®), ambos da NASA e do Departamento de Energia dos EUA, apresentou-se um cluster custando menos de $ 50.000 capaz de alcançar mais de um gigaflop por segundo mantendo a performance.

Com o rápido avanço e crescente disponibilidade de tecnologias de microprocessadores, de rede de interconexões de alta velocidade, e de outros componentes relacionados, os clusters Beowulf tornaram-se a escolha mais comum para a construção de computação paralela de alta performane.

Muitos supercomputadores comerciais usam os mesmos processadores, memórias e chips controladores que são utilizados pelos clusters Beowulf. Entretanto, clusters Beowulf utilizam somente componentes do mercado de consumo de massa que não estão sujeitos a atrasos devido a custos de peças personalizadas e de projetos exclusivos.

Um cluster Beowulf utiliza uma arquitetura multicomputacional, conforme apresentado na Fig. 4.1. Ele possui um sistema de computação paralela, que geralmente é composto por: um ou mais nós mestres; de um ou mais nós de computação, ou de nós clientes, interconectados através de uma rede de interconexões amplamente disponível. Todos os nós em um cluster Beowulf são sistemas facilmente encontrados no mercado como PCs, estações de trabalho ou servidores, em execução em softwares conhecidos, tais como o Linux e o Windows.

O nó mestre age como um servidor de Network File System (NFS), é como uma porta para o mundo exterior. Como um servidor NFS, o nó principal fornece espaço de arquivo ao usuário e outro software de sistema comum para os nós clientes via NFS.

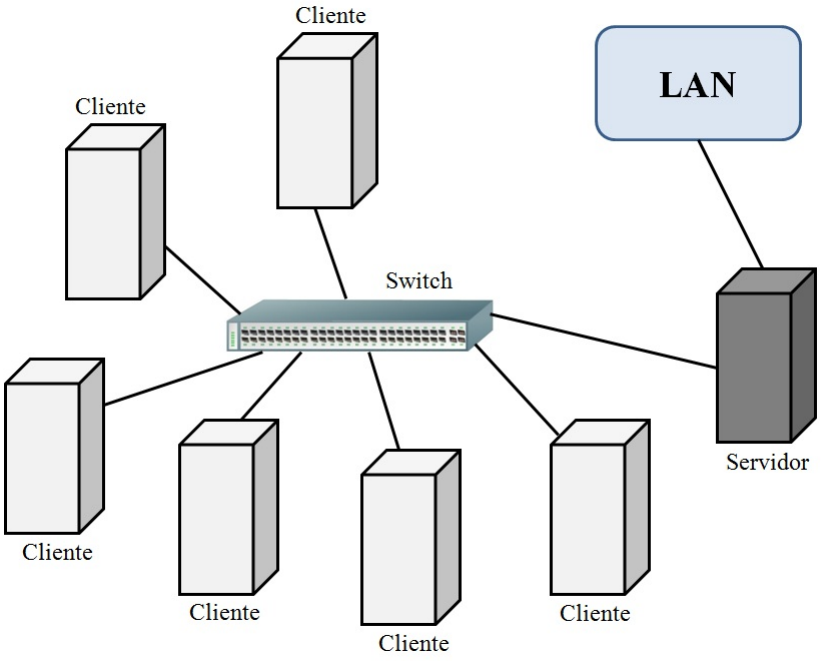


Figura 4.1 – Visão Lógica de uma Classe de Cluster Beowulf

Como um gateway, o nó mestre permite aos usuários ter acesso através dele aos nós clientes. Normalmente, o nó mestre é a única máquina que também está conectada com o mundo exterior através de uma rede por meio de uma placa de interface (NIC).

A tarefa exclusiva dos nós clientes é executar tarefas em paralelo. Na maioria dos casos, portanto, os nós de computação não têm teclados, mouses, placas de vídeo ou monitores. Todo o acesso aos nós do cliente é fornecida através de conexões remotas a partir do nó mestre. Como os nós de computação não precisam acessar máquinas de fora do cluster, nem máquinas de fora do cluster precisam acessar diretamente os nós de computação, normalmente os nós de computação usam endereços de IP privados, como as faixas de endereços 10.0.0.0 / 8 ou 192.168.0.0/16.

Do ponto de vista de um usuário, um cluster Beowulf aparece como um Processador Massivamente Paralelo (PMP) do sistema. Os métodos mais comuns de usar o sistema são: acessar o nó mestre diretamente ou através de Telnet; ou obter um login remoto a partir de estações de trabalho pessoais. Após acessar o nó mestre, os usuários podem preparar e compilar suas aplicações paralelas, e também gerar trabalho em um número desejado de nós de computação em cluster.

As aplicações devem ser escritas em paralelo e usar um programa modelo

de transmissão de mensagens. Trabalhos de aplicação paralela são gerados em nós de computação, que trabalham colaborativamente até terminarem a aplicação. Durante a execução, os nós de computação usam um padrão de transmissão de mensagens middleware, como o Message Passing Interface (MPI) e Parallel Virtual Machine (PVM), para trocar informações.

Clusters Beowulf oferecem uma série de benefícios específicos:

Custo-benefício: Uma das principais vantagens de uma cluster Beowulf é o seu custo-efetividade. Clusters Beowulf são construídos com componentes que são relativamente baratos e amplamente disponíveis no mercado.

Mantém o passo com as tecnologias: Já que os clusters Beowulf utilizam apenas os componentes de mercado de massa, é fácil de empregar as últimas tecnologias para manter o cluster no estado da arte.

Configuração flexível: Os usuários podem personalizar uma configuração que lhe é viável e repartir o orçamento com sabedoria para satisfazer as exigências de desempenho de seus aplicativos. Por exemplo, aplicações em paralelo de grão fino (que trocam muitas mensagens entre os processadores frequentemente) podem motivar os usuários a atribuir uma parcela maior do seu orçamento para interconexões de alta velocidade.

Escalabilidade: Quando o poder de processamento requer aumento, a performance e o tamanho de um cluster Beowulf pode ser facilmente expandido pela adição de mais nós de computação.

Alta disponibilidade: Cada nó de computação de um cluster Beowulf é uma máquina individual. A falha de um nó de computação não afetará os outros nós ou a disponibilidade de todo o cluster.

Compatibilidade e portabilidade: Graças à padronização e ampla disponibilidade de interfaces de passagem de mensagens, como MPI e PVM, a maioria das aplicações paralelas usam estes middlewares padrões. Uma aplicação paralela usando MPI pode ser facilmente configurada a partir da IBM® RS/6000 SP2® ou Cray® T3E a um cluster Beowulf. É por isso que as classes de clusters Beowulf estão substituindo rapidamente os caros computadores paralelos do mercado de baixo porte para o mercado

de médio porte da computação de alto desempenho.

4.2.3 Opções de Projeto para construção de um cluster Beowulf

Pode-se facilmente construir um cluster Beowulf a partir dos componentes considerados mais adequados. Não existe um cluster Beowulf que seja geral o suficiente para satisfazer as necessidades de todos, porém existem inúmeras opções de projeto para a criação de um cluster Beowulf.

A Fig. 4.2 mostra a arquitetura de um típico cluster Beowulf, destacando algumas das escolhas de projeto que podem ser consideradas.

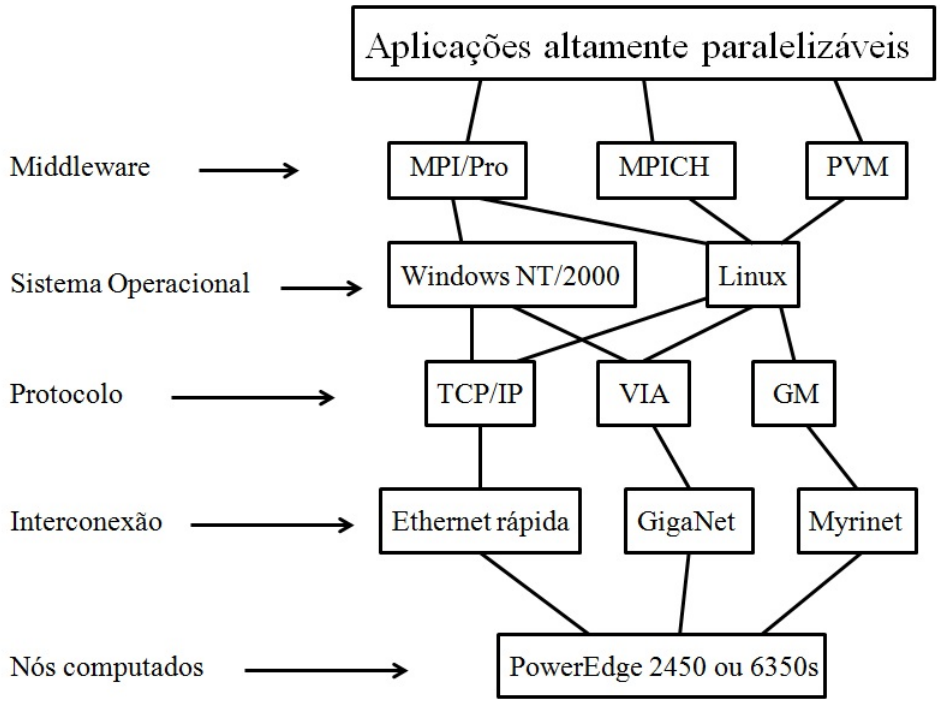


Figura 4.2 – Arquitetura de um Cluster Beowulf. Fonte: tradução de HSIEH (2000).

Nós de computação: As qualidades que devem ser consideradas para nós computacionais incluem o tipo do processador, o número de processadores por nó, tamanho e velocidade do nível 2 (L2) cache, velocidade do front-side bus, projeto do subsistema de memória e velocidade do barramento PCI.

Interconexões e protocolos de comunicação: a menos que o aplicativo possa ser muito bem particionado e apenas níveis modestos de comunicação sejam necessários, interconexões de alta velocidade desvendam o potencial de desempenho de um cluster Beowulf. Tem-se observado que as interconexões de alta velocidade, tais

como Giganet® e Myrinet, não só facilitam um maior desempenho do cluster de Fast Ethernet e Gigabit Ethernet, mas também conseguem uma melhor relação entre custo e desempenho.

Sistemas operacionais: A maioria dos clusters Beowulf são baseados em Linux. Encontra-se também outros usuários que aproveitam os recursos do Windows NT e Windows 2000 para a construção de clusters Beowulf.

Bibliotecas de passagem de mensagens (MPI): Algumas das implementações MPI usam um esquema de votação para atingir a latência de comunicação ultra-low, outros minimizam os ciclos de CPU utilizada por tarefas de comunicação com uma abordagem baseada em interrupções.

Ferramentas de desenvolvimento de aplicações: Muitas aplicações paralelas, com intensivas simulações numéricas, podem obter um melhor desempenho usando opções de otimização adequada dos compiladores e otimizando subrotinas comuns.

4.3 Métricas de Desempenho

Medir o desempenho de um processamento em paralelo é encontrar uma forma de caracterizar o comportamento do sistema para então interpretar de forma correta os resultados, e esta interpretação dependerá fortemente da escolha das métricas de aferição de desempenho. Aqui, os termos performance ou desempenho de um sistema se referirão ao tempo total de execução de um programa, desta forma quanto menor for o tempo de execução, maior será o desempenho. Em outras palavras pode-se dizer que desempenho é a capacidade de reduzir o tempo de resolução do problema à medida que os recursos computacionais aumentam.

A aferição do desempenho é muito importante pois no processo de desenvolvimento de um sistema computacional um dos principais objetivos é obter a maior eficiência com o menor custo possível.

Pode-se dividir as métricas de desempenho em duas classes: (a) nas métricas de desempenho para processadores, as quais permitem avaliar a performance de um

processador tendo por base a velocidade/número de operações que este consegue realizar num determinado espaço temporal; (b) métricas de desempenho para aplicações paralelas as quais permitem avaliar a performance de uma aplicação paralela tendo por base a comparação entre a execução com múltiplos processadores e a execução com um só processador. Este último caso será considerado em seguida.

O tempo total de execução de um programa pode ser definido como o tempo que decorre desde que o primeiro processador inicia a execução até o último processador terminar e pode ser decomposto no tempo de computação, de comunicação e no tempo ocioso, ou seja:

$$T_{exec} = T_{comp} + T_{comu} + T_{ocioso},$$

sendo que, o tempo de computação é o tempo gasto no processamento excluindo-se o tempo de comunicação e o tempo ocioso; o tempo ocioso é o período que um processador fica sem tarefas, talvez, por exemplo, esperando por algum dado de outro processador, o que pode ser minimizado com uma distribuição de carga adequada ou sobrepondo a computação com a comunicação e finalmente, o tempo de comunicação é o tempo que o algoritmo gasta para enviar e receber mensagens.

De acordo com QUENTAL (2006), “desempenho em programação paralela é uma questão multifacetada. Além do tempo de execução e escalabilidade é importante considerar os mecanismos pelos quais os dados são gerados, armazenados, transmitidos na rede, transferidos de/para o disco e transportados nos diferentes estágios de computação. As principais métricas encontradas nesta área são: tempo de execução e throughput (citados anteriormente), requisitos de memória, requisitos de entrada/saída, latência (tempo que uma unidade de informação leva para chegar ao seu destino através de um meio de comunicação), custos de projeto, de implementação, verificação de hardware, de potencial de reuso e de portabilidade”. A comunicação e a sincronização entre diferentes subtarefas é tipicamente uma das maiores barreiras para atingir um grande desempenho em programas paralelos.

Existem várias métricas que permitem medir e/ou avaliar o desempenho de uma aplicação paralela. Aqui serão apresentadas as mais conhecidas.

Parte do texto seguinte foi extraído de ROCHA (2007).

As métricas *Speedup* (S_p) e Eficiência (E_p) são equações das frações de tempo que os processadores passam realizando trabalho útil, caracterizando a efetividade de se utilizar paralelismo em relação a uma versão sequencial.

Sejam p o número de processadores, T_1 o tempo em segundos de execução do programa em um processador e T_p o tempo em segundos de execução do programa em paralelo com p processadores. O *Speedup*, dado pela Eq. (4.1), é a razão entre o tempo de execução sequencial e o tempo de execução em paralelo e medirá o grau de desempenho do processamento, ou seja, o ganho de velocidade de processamento.

$$S_p = \frac{T_1}{T_p} \tag{4.1}$$

O *speedup linear* ou *speedup ideal* é dado por $S_{p_l} = p$, isto é, o *speedup* ideal significa que o tempo de execução do algoritmo sequencial em um processador é igual ao tempo de execução do algoritmo em paralelo com p processadores multiplicado por p, em outras palavras, $T_1 = pT_p$, o que indica que toda a capacidade computacional disponível no sistema foi utilizada para ganhar velocidade na execução do algoritmo. Mas tal valor é quase impossível de se alcançar, pois para que isto aconteça o algoritmo precisa ser altamente paralelizável e a comunicação entre os processadores deve ser mínima.

Alguns fatores que podem gerar sobrecargas no sistema e portanto impedir que o sistema atinja o valor do *speedup* ideal são o excesso de comunicação entre os processadores, o nivel de paralelismo utilizado, distribuição de carga e a existência de partes do código executável que sejam estritamente sequenciais.

Se a sobrecarga da paralelização for muito alta então $T_1 < T_p$ donde $S_p < 1$. Esta situação indesejável é chamada de *slowdown*. O comportamento comum do *speedup* é chamado de sublinear e ocorre quando $1 < S_p < p$. O *speedup ideal* é dado por $S_{p_l} = p$ e *speedup* supralinear ocorre quando $S_p > p$, não é uma situação comum mas é possível de acontecer.

Por outro lado, a Eficiência, dada pela Eq. (4.2), é a razão entre o grau de desempenho e a quantidade de recursos computacionais e medirá o grau de aproveita-

mento destes recursos disponíveis.

$$E_p = \frac{S_p}{p} = \frac{T_1}{pT_p} \tag{4.2}$$

Observe que algoritmos com *speedup ideal* ou algoritmos sendo executados em um único processador tem eficiência igual a um ou 100% porém, a medida que o número de processadores vai aumentando a eficiência se aproxima de zero visto que $\lim_{p\to\infty} \frac{T_1}{pT_p} = 0$.

De forma similar à classificação do *speedup* pode-se classificar a eficiência. Se $E_p < 1$ o sistema encontra-se na situação de slowdown; se $E_p < 1$, então é sublinear, se $E_p = 1$ é linear e se $E_p > 1$ então é supralinear.

Pelo que foi visto pode-se concluir que a eficiência de uma aplicação é uma função decrescente em relação ao número de processadores, Fig. 4.3, ou é uma função crescente em relação ao tamanho do problema, Fig. 4.4.

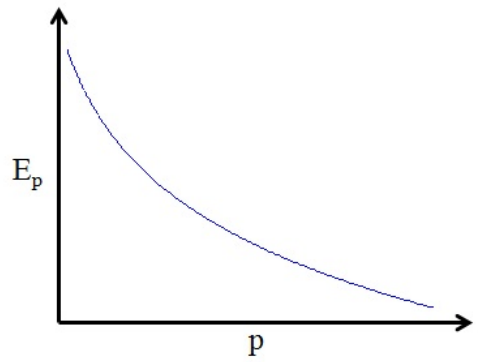


Figura 4.3 – Eficiência para tamanho de problema fixo

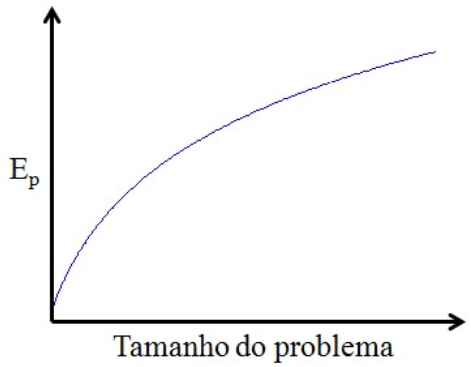


Figura 4.4 – Eficiência com número de processadores fixo

Uma aplicação é dita escalável quando demonstra a capacidade de manter a mesma eficiência à medida que o número de processadores e a complexidade do problema aumentam proporcionalmente.

É importante lembrar que numa aplicação sempre existe uma parte do algoritmo que não pode ser paralelizada, ou seja, sempre existem partes do algoritmo que são estritamente sequenciais. Afim de se obter um limite máximo do valor do *speedup* que pode-se encontrar ao resolver um determinado problema com p processadores será considerado a Lei de Amdahl. Seja s a fração do código que é estritamente sequencial

assim, $(1 - s)$ será a parte suscetível de ser paralelizada. Observe que na Lei de Amdahl parte do tempo de execução sequencial é utilizado para estimar o speedup máximo possível de ser alcançado em múltiplos processadores. As Figs. 4.5 e 4.6 ilustram a relação entre tempo de execução e número de processadores na visão de Amdahl, na qual assume-se que s seja constante para qualquer quantidade de processadores.

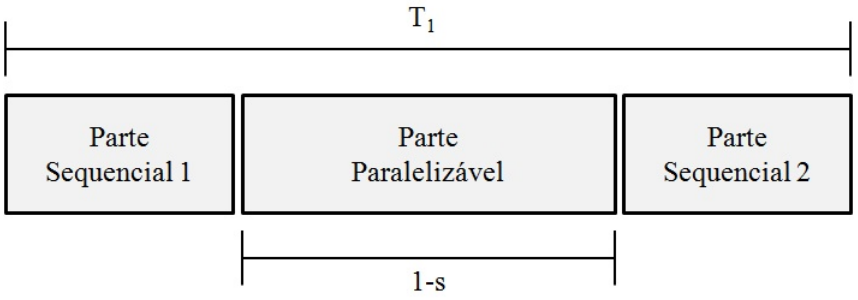


Figura 4.5 – Lei de Amdahl: tempo de execução em um único processador

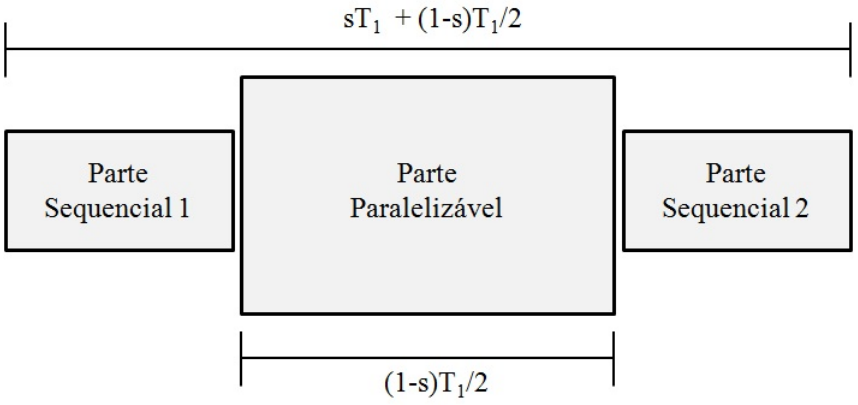


Figura 4.6 – Lei de Amdahl: tempo de execução em dois processadores

Além disso, a computação realizada por uma aplicação paralela pode ser dividida em computações que só podem ser realizadas sequencialmente, C_{seq}, em computações que podem ser realizadas em paralelo, C_{par}, e em computações de comunicação/sincronização/iniciação, C_{com}. Desta forma o *speedup* pode ser reescrito da seguinte maneira:

$$S_p = \frac{T_1}{T_p} = \frac{C_{seq} + C_{par}}{C_{seq} + \frac{C_{par}}{p} + C_{com}} \tag{4.3}$$

Como, em aplicações paralelas, $C_{com} \geq 0$ segue que $C_{seq} + \frac{C_{par}}{p} \leq C_{seq} + \frac{C_{par}}{p} +$

C_{com}, donde

$$\frac{C_{seq}+C_{par}}{C_{seq}+\frac{C_{par}}{p}} \geq \frac{C_{seq}+C_{par}}{C_{seq}+\frac{C_{par}}{p}+C_{com}} = S_p \quad \rightarrow \quad S_p \leq \frac{C_{seq}+C_{par}}{C_{seq}+\frac{C_{par}}{p}} \tag{4.4}$$

Como s representa a fração da computação que só pode ser realizada sequencialmente segue que $s = \frac{C_{seq}}{C_{seq}+\frac{C_{par}}{p}}$, substituindo na Eq. (4.4) e simplificando o resultado chega-se na Lei de Amdahl:

$$S_p \leq \frac{1}{s-\frac{1-s}{p}} \qquad 0 \leq s \leq 1 \tag{4.5}$$

Pode-se também utilizar a lei de Amdahl para determinar o limite máximo de *speedup* que uma determinada aplicação poderá alcançar independentemente do número de processadores a utilizar, isto é,

$$\lim_{p\to\infty} \frac{1}{s-\frac{1-s}{p}} = \frac{1}{s} \tag{4.6}$$

Observe que a lei de Amdahl ignora o custo das operações de comunicação/sincronização associadas à introdução de paralelismo numa aplicação. Por esse motivo, a lei de Amdahl pode resultar em predições pouco realísticas para determinados problemas. Mais ainda, à medida que se aumenta o parâmetro p o *Speedup*, segundo a Lei de Amdahl, estará limitado a $1/s$, o que tornou-se um incômodo para os fabricantes de máquinas de grande porte. Por este motivo, uma alternativa para se medir o desempenho do sistema é a utilização da Lei de Gustafson-Barsis. Esta lei parte do princípio de que todo problema suficientemente grande pode ser eficientemente paralelizável, diferentemente da Lei de Amdahl no qual a fração que pode ser executada em paralelo e a quantidade de processadores dependem um do outro. Mais ainda, como será visto, pela Lei de Gustafson-Barsis o *Speedup* cresce indefinidamente.

Afim de se estruturar a Lei de Gustafson-Barsis considere novamente a Eq. (4.4). Como

$$s = \frac{C_{seq}}{C_{seq}+\frac{C_{par}}{p}} \quad \text{segue que} \quad 1-s = \frac{\frac{C_{par}}{p}}{C_{seq}+\frac{C_{par}}{p}} \tag{4.7}$$

Assim, tem-se que

$$C_{seq} = s\left(C_{seq} + \frac{C_{par}}{p}\right) \quad \text{e} \quad C_{par} = p(1-s)\left(C_{seq} + \frac{C_{par}}{p}\right) \tag{4.8}$$

Substituindo na Eq. (4.4) segue

$$S_p \leq \frac{C_{seq} + C_{par}}{C_{seq} + \frac{C_{par}}{p}} = \frac{s\left(C_{seq} + \frac{C_{par}}{p}\right) + p(1-s)\left(C_{seq} + \frac{C_{par}}{p}\right)}{C_{seq} + \frac{C_{par}}{p}} = \frac{(s + p(1-s))\left(C_{seq} + \frac{C_{par}}{p}\right)}{\left(C_{seq} + \frac{C_{par}}{p}\right)} \tag{4.9}$$

Logo, a Lei de Gustafson-Barsis é dada pela inequação

$$S_p \leq p + s(1-p) \tag{4.10}$$

Como analisado, a Lei de Gustafson-Barsis parte do tempo de execução em paralelo para estimar o *speedup* máximo comparado com a execução sequencial. Além disso, *speedup* máximo segundo a Lei de Gustafson-Barsis converge para o infinito quando o número de processadores cresce indefinidamente.

Mas, assim como na lei de Amdahl, a Lei de de Gustafson-Barsis também tem limitações, ao usar o tempo de execução em paralelo como o ponto de partida, em lugar do tempo de execução sequencial, a Lei de Gustafson-Barsis assume que a execução com um só processador é no pior dos casos p vezes mais lenta que a execução com p processadores o que pode não ser verdade.

Finalmente, será abordado aqui a Métrica de de Karp-Flatt. Seja $\alpha = \frac{C_{seq}}{T_1}$ a razão sequencial determinada experimentalmente numa computação paralela. Assim, $C_{seq} = \alpha T_1$ e lembrando que se pode escrever $T_1 = C_{seq} + C_{par}/p$ segue que $C_{par} = T_1 - C_{seq} = T_1 - \alpha T_1 = (1-\alpha)T_1$. Considerando que C_{com} é desprezável em $T_p = C_{seq} + C_{par}/p + C_{com}$ vem que

$$T_p = \alpha T_1 + \frac{(1-\alpha)T_1}{p} \tag{4.11}$$

Mas, $S_p = \frac{T_1}{T_p} \quad \rightarrow \quad T_1 = S_P T_p$. Substituindo na Eq. (4.11) segue que:

$$T_p = \alpha S_p T_p + \frac{(1-\alpha) S_p T_P}{p}, \tag{4.12}$$

simplificando T_p em ambos os lados da equação obtem-se

$$1 = \alpha S_p + \frac{(1-\alpha) S_p}{p} = S_p \left(\alpha + \frac{1-\alpha}{p} \right) \quad \longrightarrow \quad \frac{1}{S_p} = \alpha + \frac{1-\alpha}{p} = \alpha \left(1 - \frac{1}{p} \right) + \frac{1}{p}$$

ou seja,

$$\frac{1}{S_p} = \alpha \left(1 - \frac{1}{p} \right) + \frac{1}{p} \tag{4.13}$$

Isolando α chega-se a Métrica de de Karp-Flatt:

$$\alpha = \frac{\frac{1}{S_p} - \frac{1}{p}}{1 - \frac{1}{p}} \tag{4.14}$$

Observe que, por definição, α é um valor constante que não depende do número de processadores, mas pela métrica de Karp-Flatt α é uma função do número de processadores. Como a eficiência duma aplicação é uma função decrescente do número de processadores, a métrica de Karp-Flatt permite-nos determinar qual a importância da componente C_{com} nesse decréscimo.

Se os valores de α forem constantes à medida que o número de processadores aumenta isso significa que a componente C_{com} é também constante. Logo, o decréscimo da eficiência é devido à existência de pouco paralelismo no problema.

Por outro lado, se os valores de α aumentarem à medida que o número de processadores aumentam isso significa que o decréscimo é devido à componente C_{com}, ou seja, à existência de custos excessivos associados à computação em paralelo (custos de comunicação, sincronização e/ou iniciação da computação).

Há ainda várias métricas para se avaliar o desempenho de um processo em paralelo que não serão analizadas aqui. Para mais considerações de métricas de desempenho em computação paralela pode-se citar os livros HWANG (1993) e STALLINGS (2010).

Afim de ilustrar o uso das métricas *speedup* e a eficiência, suponha que o tempo em segundos de execução em um mesmo problema de 1, 2, 4, 8 e 16 processadores sejam respectivamente, 1500, 850, 470, 260 e 150. A Tab. 4.1 apresenta os valores obtidos do *speedup* e da eficiência para cada caso. Note que estas métricas são grandezas adimencionais.

Tabela 4.1 – Ilustração sobre as métricas *Speedup* e Eficiência

	1 CPU	2CPUs	4 CPUs	8 CPUs	16 CPUs
T_p	1500	850	470	260	150
S_p	$S_1 = \frac{1500}{1500} = 1$	$S_2 = \frac{1500}{850} = 1,76$	$S_4 = \frac{1500}{470} = 3,19$	$S_8 = \frac{1500}{260} = 5,77$	$S_{16} = \frac{1500}{150} = 10$
E_p	$E_1 = \frac{1}{1} = 1$	$E_2 = \frac{1,76}{2} = 0,88$	$E_4 = \frac{3,19}{4} = 0,80$	$E_8 = \frac{5,77}{8} = 0,72$	$E_{16} = \frac{10}{16} = 0,62$

No exemplo ilustrativo apresentado na Tab. 4.1 o *speedup* em todos os casos é sublinear, isto é, $1 < S_p < p$. Como previsto na Lei de Gustafson-Barsis o *speedup* aumenta quando o número de processadores cresce. A eficiência neste exemplo também é sublinear e decrescente com relação ao número de processadores.

CAPÍTULO V

SIMULAÇÕES NUMÉRICAS

Este capítulo é voltado para as simulações numéricas realizadas durante o estudo a fim de comprovar a eficiência dos algoritmos. Foram realizadas diversas simulações com Algoritmo Genético, Evolução Diferencial, Evolução com Conjuntos Embaralhados e Evolução Diferencial Melhorada com Processamento em Paralelo.

O Toolbox de Otimização Algoritmos Genéticos (GAOT), foi desenvolvido por HOUCK; JOINEZ; KAY (1985). O código computacional da ED foi desenvolvido em MATLAB® pela autora com a coloboração dos orientadores desta tese. O código computacional da ECE foi desenvolvido em MATLAB® por DUAN (2004). O algoritmo da EDMP foi desenvolvido em C++ pela autora com a coloboração do Professor Doutor José Laércio Doricio.

Nas próximas seções serão apresentados os resultados encontrados. Visto que as simulações numéricas ocorreram em momentos diferentes, em cada seção será mencionado o modelo e as configurações do computador utilizado.

5.1 Simulações com os algoritmos AG, ED e ECE

O objetivo desta seção é avaliar a performance do método Evolução com Conjuntos Embaralhados a fim de analisar a velocidade de execução do método. Para testar e avaliar o método ECE, serão propostos, a seguir, alguns problemas teste necessários

para comparar o método com os Algoritmos Genéticos e Evolução Diferencial ainda não modificado.

Os parâmetros utilizados em ED foram: número de indivíduos na população N_p = 20; 200 gerações, fator de diferença ponderada F = 0,8 e probabilidade de cruzamento CR = 0,6.

Foi adotado para executar AG N_p = 80 indivíduos, 100 gerações, probabilidades de cruzamento e mutação: 0,60 e 0,02, respectivamente.

Para ECE foi adotado $9 = 2 \times 4 + 1$ indivíduos na população, onde 4 é o número de variáveis de projeto.

As simulações numéricas desta seção 5.1 foram desenvolvidas em um processador Intel® Core^{TM} 2 Duo T5450. Note que as unidades das medidas dos resultados apresentados nas subseções 5.1.1 e 5.1.2 não estão no Sistema Internacional de Unidades. A fim de facilitar a comparação dos resultados obtidos, foi adotado o Sistema Inglês de unidade, o mesmo sistema de medidas adotado pelas referências (COELHO (2000), DEB; DASGUPTA; MICHALEWICZ (1997) e TEIXEIRA et al (2011)).

5.1.1 Projeto de um recipiente de pressão

O objetivo deste problema é o de minimizar o custo total, incluindo o custo do material, modelagem e soldagem, do projeto de um recipiente cilíndrico que é limitado em suas extremidades por tampas hemisféricas conforme proposto por COELHO (2000), DEB; DASGUPTA; MICHALEWICZ (1997) e TEIXEIRA et al (2011) e apresentado na Fig. 5.1.

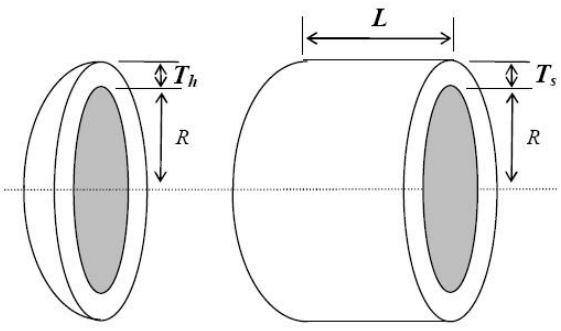


Figura 5.1 – Projeto de um recipiente de pressão.

Existem quatro variáveis no projeto: x_1 (T_s, espessura da casca), x_2 (T_h, espessura da tampa), x_3 (R, raio interno) e x_4 (L, comprimento da secção cilíndrica do recipiente), não incluindo a tampa. Além disso, os valores das variáveis x_1 e x_2 são múltiplos de 0,0625 polegadas, pois referem-se às espessuras de chapas de aço laminadas padronizadas, e $x_3, x_4 \in \mathbb{R}$. O problema pode ser descrito conforme segue:

$$\text{min } f(x) = 0,6224x_1x_3x_4 + 1,7781x_2x_3^2 + 3,1661x_1^2x_4 + 19,84x_1^2x_3 \tag{5.1}$$

Sujeito às seguintes restrições:

$$g_1(x) = -x_1 + 0,0193x_3 \leq 0 \tag{5.2}$$

$$g_2(x) = -x_2 + 0,00954x_3 \leq 0 \tag{5.3}$$

$$g_3(x) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0 \tag{5.4}$$

$$g_4(x) = x_4 - 240 \leq 0 \tag{5.5}$$

Os limites laterais das variáveis $x = (x_1, x_2, x_3, x_4)$ utilizados foram: $0 \leq x_1, x_2 \leq 10$, $10 \leq x_3 \leq 100$, $100 \leq x_4 \leq 200$.

Na Tab. 5.1 são apresentados alguns valores encontrados na literatura e a solução obtida utilizando Algoritmos Genético e Evolução Diferencial para a resolução do problema de recipiente de pressão. A Tab. 5.2 mostra os resultados obtidos com o método Evolução com Conjuntos Embaralhados considerando quantidades diferentes de conjuntos, sendo que t representa o tempo de processamento em segundos. Observe que os menores valores para a função objetivo foram obtidos com o método Evolução com Conjuntos Embaralhados e que além disso, independente do número de conjuntos, todas as restrições foram obedecidas no ponto ótimo, que é de fundamental importância para projetos de engenharia.

Tabela 5.1 – Comparação de alguns resultados do problema de um recipiente de pressão.

Variáveis de projeto	Hu *et al.* (2003)	Coello (2000)	Deb (1997)	Algoritmo Genético t=19,23s	Evolução Diferencial t=18,54s
$x_1(T_s)$	0,8125	0,8125	0,9375	0,9613	0,2233
$x_2(T_h)$	0,4375	0,4375	0,5000	0,4755	0,9611
$x_3(R)$	42,09845	40,3239	48,3290	49,7928	53,7741
$x_4(L)$	176,6366	200,0000	112,6790	100,0000	175,9351
$g_1(x)$	0,0	-0,034324	-0,004750	-0,0003	0,8146
$g_2(x)$	-0,03588	-0,052847	-0,038941	-0,0005	-0,4481
$g_3(x)$	-5,820e-11	-27,10584	-3652,876	-21,0985	-953602,7787
$g_4(x)$	-63,3624	-40,0000	-127,3210	-140,0000	-64,0649
$f(x)$	6059,1312	6288,7445	6410,3811	6281,3716	6343,9254

Tabela 5.2 – Resultados do problema de um recipiente de pressão para ECE.

Variáveis de projeto	2 Conjuntos t=5,26s	4 Conjuntos t=5,72s	10 Conjuntos t=5,54s	20 Conjuntos t=7,46s
$x_1(T_s)$	0,8631	0,8555	0,7787	0,7784
$x_2(T_h)$	0,4267	0,4229	0,3850	0,3842
$x_3(R)$	44,6980	44,3239	40,3499	40,3275
$x_4(L)$	146,9913	150,8823	199,5795	199,8933
$g_1(x)$	-0,0004	-9,15e-6	-6,72e-7	-4,0e-5
$g_2(x)$	-0,0003	-6,14e-5	-8,26e-5	-1,0e-4
$g_3(x)$	-679,6647	-3,3706	-7,3757	-13,1134
$g_4(x)$	-93,0087	-89,1177	-40,4205	-40,1067
$f(x)$	6052,4298	6031,2781	5886,6091	5886,2353

5.1.2 Projeto de uma viga engastada

O objetivo é minimizar o custo de uma viga de aço com as limitações de tensão de cisalhamento, esforço de flexão na viga, flambagem de carga na barra, deflexão do feixe de extremidade da viga e restrições laterais, conforme proposto por COELHO (2000), DEB; DASGUPTA; MICHALEWICZ (1997) e TEIXEIRA et al (2011) e esquematizado na Fig. 5.2. Além disso, existem quatro variáveis de projeto: x_1 (h, espessura da solda), x_2 (l, comprimento da junta soldada), x_3 (t, largura do feixe) e x_4 (b, espessura da viga).

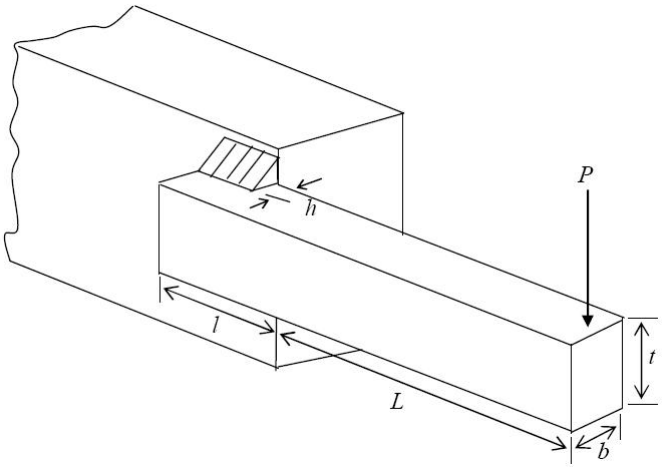


Figura 5.2 – Projeto de uma viga engastada.

O problema pode ser expresso como:

$$min\ f(x) = 1,10471x_1^2x_2 + 0,04811x_3x_4(14,0 + x_2) \tag{5.6}$$

Sujeito às seguintes restrições:

$$g_1(x) = \tau(x) - \tau_{max} \leq 0 \tag{5.7}$$

$$g_2(x) = \sigma(x) - \sigma_{max} \leq 0 \tag{5.8}$$

$$g_3(x) = x_1 - x_4 \leq 0 \tag{5.9}$$

$$g_4(x) = 0,10471x_1^2 + 0,04811x_3x_4(14,0 + x_2) - 5 \leq 0 \tag{5.10}$$

$$g_5(x) = 0,125 - x_1 \leq 0 \tag{5.11}$$

$$g_6(x) = \delta(x) - \delta_{max} \leq 0 \tag{5.12}$$

$$g_7(x) = P - P_c(x) \leq 0 \tag{5.13}$$

sendo que,

$$\tau(x) = \sqrt{\tau'^2 + 2\tau'\tau'' \frac{x_2}{2R} + \tau''^2} \tag{5.14}$$

$$\tau' = \frac{P}{\sqrt{2}x_1x_2} \tag{5.15}$$

$$\tau'' = \frac{MR}{J} \tag{5.16}$$

$$M = P(L + \frac{x_2}{2}) \tag{5.17}$$

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} \tag{5.18}$$

$$J = 2\left[\sqrt{2}x_1x_2\left(\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right)\right] \tag{5.19}$$

$$\sigma(x) = \frac{6PL}{x_4x_3^2} \tag{5.20}$$

$$\delta(x) = \frac{4PL^3}{Ex_3^3x_4} \tag{5.21}$$

$$P_c(x) = \frac{4,013E\sqrt{\frac{x_3^2x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right) \tag{5.22}$$

onde:

$P = 6000\ lb$, $L = 14\ in$, $E = 30 \cdot 10^6\ psi$, $G = 12 \cdot 10^6\ psi$, $\tau_{max} = 13600\ psi$, $\sigma_{max} = 30000\ psi$ e $\delta_{max} = 0,25\ in$.

Os seguintes limites laterais das variáveis $x = (x_1, x_2, x_3, x_4)$ forão adotados: $0,1\ in \leq x_1, x_4 \leq 2\ in$ e $0,1\ in \leq x_2, x_3 \leq 10\ in$.

Na Tab. 5.3 são exibidos os resultados encontrados na literatura e obtidos com os algoritmos Genético e Evolução Diferencial. Na Tab. 5.4 são apresentados os valores ótimos calculados pelo método Evolução com Conjuntos Embaralhados. De forma similar, t representa o tempo de processamento em segundos.

Tabela 5.3 – Comparação de alguns resultados para o projeto de uma viga engastada.

Variáveis de projeto	Hu *et al.* (2003)	Coello (2000)	Deb (1997)	Algoritmo Genético t=19,65s	Evolução Diferencial t=18,98s
$x_1(h)$	0,20573	0,20880	0,2489	0,409904	0,190675
$x_2(l)$	3,47049	3,42050	6,1730	2,015463	4,219093
$x_3(t)$	0,03662	8,99750	8,1739	6,533373	8,466207
$x_4(b)$	0,20573	0,21000	0,2533	0,393606	0,240833
$g_1(x)$	0,0	-0,337812	-5758,603	-0,049774	-299,421047
$g_2(x)$	0,0	-353,9026	-255,5769	-1,930750	-803,114581
$g_3(x)$	-5,551e-17	-0,001200	-0,004400	0,016298	-0,050158
$g_4(x)$	-3,432983	-3,411865	-2,982866	-3,000997	-3,209018
$g_5(x)$	-0,080729	-0,083800	-0,123900	-0,284904	-0,065675
$g_6(x)$	-0,235540	-0,235649	-0234160	-0,230001	-0,234979
$g_7(x)$	-9,094e-13	-363,2323	-4465,270	-27262,031799	-3212,606384
$f(x)$	1,7248508	1,7483094	2,433116	2,358168	1,956630

Tabela 5.4 – Resultados do problema de uma viga engastada para ECE.

Variáveis de projeto	2 Conjuntos t=7,41s	4 Conjuntos t=4,88s	10 Conjuntos t=6,72s	20 Conjuntos t=7,99s
$x_1(h)$	0,2794	0,2055	0,2057	0,2058
$x_2(l)$	2,8417	3,4753	3,4724	3,4703
$x_3(t)$	7,4943	9,0393	9,0366	9,0358
$x_4(b)$	0,2991	0,2057	0,2058	0,2058
$g_1(x)$	-0,5225	-2,1166	-3,9511	-1,1344
$g_2(x)$	-0,8868	-17,8126	-4,2678	-0,4141
$g_3(x)$	-0,0197	-0,0002	-0,0001	0
$g_4(x)$	-3,1754	-3,4321	-3,4326	-3,4328
$g_5(x)$	-0,1544	-0,0805	-0,0807	-0,0808
$g_5(x)$	-0,2326	-0,2355	-0,2355	-0,2355
$g_7(x)$	-1089,7967	-1,1995	-2,6463	-3,1720
$f(x)$	2,0614	1,7256	1,7253	1,7251

Assim como HU; EBERHART; SHI (2003), os menores valores para a função objetivo foram obtidos com o método Evolução com Conjuntos Embaralhados. Além disso, independente do número de conjuntos, todas as restrições foram obedecidas no ponto ótimo.

Os resultados obtidos com o método da Evolução com Conjuntos Embaralhados mostram que este foi eficiente em encontrar um valor ótimo para os problemas apresentados, principalmente quando se aumenta o número de conjuntos nos quais o algoritmo busca um valor ótimo. Embora os problemas estudados sejam relativamente simples de serem resolvidos, enquanto os Algoritmos Genéticos e Evolução Diferencial

levaram cerca de 20 segundos para obter o ótimo, o método Evolução com Conjuntos Embaralhados precisou, em média, de um tempo computacional três vezes menor.

Estes testes iniciais com ECE confirmaram as intuições de que unir em um único algoritmo a simplicidade e robustez da ED com os conjuntos embaralhados da ECE poderiam dar certo. Ainda, como um dos objetivos é diminuir o tempo de processamento o novo algoritmo Evolução Diferencial Melhorada foi implementado em paralelo. A próxima seção apresentará os resultados dos primeiros testes realizados com EDMP.

5.2 Aplicações Simples em Processamento Paralelo

Várias funções testes de otimização, irrestritas e restritas, foram utilizadas a fim de se encontrar possíveis erros de implementação e verificar a capacidade de otimização do código computacional Evolução Diferencial Melhorada implementado em paralelo. Todos os problemas usados para validação do código foram consultados no site OPTIMA (2012). As simulações foram realizadas com 1, 2, 4, 8, 16 e 32 processadores e os resultados serão apresentados a seguir.

O código computacional para EDMP foi desenvolvido em C++ pela autora com colaboração do Professor Doutor José Laércio Doricio. As simulações numéricas desta seção 5.2 foram desenvolvidas no cluster do Laboratório de Mecânica dos Fluidos e Análise Numérica da FACIP cujos 36 processadores homogêneos são AMD AM3 Phenom II FX-6100 3.3 GHz.

Em todos os problemas foram usados 1600 indivíduos na população inicial, 200 iterações do algoritmo da ED em cada execução do algoritmo da EDMP, taxa de pertubação F=0,8 e probabilidade de cruzamento CR=0,6.

Observe que a escolha do número de indivíduos está relacionado ao número de processadores visto que a quantidade de indivíduos deve ser divisível pela quantide de processadores que será utilizada nas simulações. Foi adotado 1600 indivíduos pois este número é divisível por 1, 2, 4, 8, 16 e 32 processadores. Assim, no caso das simulações com 32 processadores haverá 50 indivíduos em cada processador.

Tabela 5.5 – Função teste de Beale.

Número de processadores	Tempo em segundos	Speedup	Eficiência	$x_{\text{ótimo}}$	$f(x_{\text{ótimo}})$
1	51,5	1	1	(3; 0,5) $(3;0,5)^*$ 0^{**}	$f=0$ $\overline{f}=0^*$ $\sigma=0^{**}$
2	24,1	2,14	1,07	(3; 0,5) $(3;0,5)^*$ 0^{**}	$f=0$ $\overline{f}=0^*$ $\sigma=0^{**}$
4	12,5	4,12	1,03	(3; 0,5) $(3;0,5)^*$ 0^{**}	$f=0$ $\overline{f}=0^*$ $\sigma=0^{**}$
8	6,8	7,58	0,95	(3; 0,5) $(3;0,5)^*$ 0^{**}	$f=0$ $\overline{f}=0^*$ $\sigma=0^{**}$
16	2,8	18,39	1,15	(3; 0,5) $(3;0,5)^*$ 0^{**}	$f=0$ $\overline{f}=0^*$ $\sigma=0^{**}$
32	2,26	22,79	0,71	(3; 0,5) $(3;0,5)^*$ 0^{**}	$f=0$ $\overline{f}=0^*$ $\sigma=0^{**}$

* Valores médios e ** desvios padrões.

5.2.1 Função de Beale

Seja o problema de otimização irrestrito dado por:

$$min\ f(x) = (1,5 - x_1 + x_1x_2)^2 + (2,25 - x_1 + x_1x_2^2)^2 + (2,625 - x_1 + x_1x_2^3)^2 \qquad (5.23)$$

Considerando os limites laterais das variáveis: $-4,5 \leq x_i \leq 4,5$ i=1,2. Sabe-se que $x^* = (3;0,5)$ e $f(x^*) = 0$. Os resultados obtidos são apresentados na Tab. 5.5.

Observe que o tempo de processamento sequencial é 18 vezes maior que o tempo necessário para 16 processadores resolver o mesmo problema e que em todos os casos a solução ótima foi encontrada. No entanto, a diferença do tempo de processamento entre 16 e 32 processadores é de décimos de segundos. Um dos motivos desta pequena diferença de tempo se deve ao aumento da comunicação que resulta numa computação menos eficiente e, portanto, mais lenta.

O speedup indica quantas vezes o programa em paralelo é mais rápido que a versão sequencial para executar uma dada tarefa. Se $S_p > 1$ a versão paralela reduziu

o tempo de execução, isto é, ficou mais rápido que a sequencial. Se $S_p < 1$ a versão paralela aumentou o tempo de execução, ou seja, ficou mais lenta que a sequencial.

Nas simulações com a função de Beale a versão paralela é mais rápida que a versão sequencial já que em todos os casos tem-se $S_p > 1$. No entanto, o valor do speedup com mais de 16 processadores tende a estagnar e, desde que se mantenha o problema fixo, o tempo de execução não diminuirá muito com o aumento do número de processadores.

Visto que para 2, 4 e 16 processadores o valor do speedup é maior que o número de processadores, nestes casos o speedup mostra-se super linear. Para 8 e 32 processadores o speedup é sublinear e para 1 processador ele é linear.

Normalmente as unidades ativas ficam parte do tempo esperando por resultados vizinhos assim, o objetivo da métrica eficiência é indicar a taxa de utilização média das unidades ativas, isto é, estima-se o quão bem os processadores estão sendo utilizados para resolver o problema.

A eficiência ideal é aquela em que em 100% do tempo cada unidade está ativa. Valores acima de 1 indicam eficiência super linear como nos casos das simulações com 2, 4 e 16 processadores apresentados na Tab. 5.5. Já nas simulações com 8 e 32 processadores a eficiência mostrou-se sublinear.

5.2.2 Função de Michalewicz

Neste teste o objetivo é minimizar a função de Michalewicz para dez variáveis, isto é, n=10, cuja função objetivo é dada por:

$$min\ f(x) = -\sum_{i=1}^{n} sen(x_i)\left(sen\left(\frac{i * x_i^2}{\pi}\right)\right)^{2n} \tag{5.24}$$

sendo $0 \leq x_i \leq \pi,\ \ i = 1, \cdots, 10$ os limites laterais da função e $f(x_{\text{ótimo}}) = -9,66015$.

Os valores ótimos podem ser observados na Tab. 5.6. Para simplificar, considerando que n=10, foram omitidos os valores encontrados relacionados ao $x_{\text{ótimo}}$.

Neste teste, houve uma diminuição significativa do tempo de processamento em paralelo quando comparado ao sequencial. Por exemplo, ao utilizar 2 processado-

Tabela 5.6 – Função teste de Michalewicz.

Número de processadores	Tempo em segundos	Speedup	Eficiência	$f(x_{\text{ótimo}})$
1	100,4	1	1	$f = -9,66015$ $\overline{f} = -9,60704^{*}$ $\sigma = 0,10346^{**}$
2	46,27	2,17	1,08	$f = -9,66015$ $\overline{f} = -9,59844^{*}$ $\sigma = 0,16887^{**}$
4	23,77	4,22	1,05	$f = -9,66015$ $\overline{f} = -9,57757^{*}$ $\sigma = 0,20428^{**}$
8	12,82	7,83	0,96	$f = -9,66015$ $\overline{f} = -9,56889^{*}$ $\sigma = 0,24053^{**}$
16	6,35	15,81	0,99	$f = -9,66015$ $\overline{f} = -9,54849^{*}$ $\sigma = 0,25475^{**}$
32	4,14	24,25	0,76	$f = -9,66015$ $\overline{f} = -9,53553^{*}$ $\sigma = 0,27983^{**}$

* Valores médios e ** desvios padrões.

res na execução do programa o tempo de processamento reduziu pela metade. O que representa um excelente ganho em tempo de execução.

Embora os desvios padrões do valor ótimo da função objetivo tenha aumentado com o aumento do número de processadores, fato este justificado pelo grande número de avaliações da função objetivo, em todos os casos a solução ótima encontrada corresponde à analítica.

Pode-se observar na Tab. 5.6 que nas simulações com 2 e 4 processadores o speedup e a eficiência apresentaram valores super lineares.

5.2.3 Função de Levy

A função de Levy é dada pela seguinte equação:

$$min\ f(x) = sen^2(\pi z_1) + \sum_{i=1}^{n-1} \left[(z_i - 1)^2(1 + 10sen^2(\pi z_i + 1))\right] + (z_n - 1)^2 \left(1 + sen^2(2\pi z_n)\right) \tag{5.25}$$

Tabela 5.7 – Função teste de Levy.

Número de processadores	Tempo em segundos	Speedup	Eficiência	$f(x_{\text{ótimo}})$
1	218,67	1	1	$f = 0$ $\overline{f} = 0,00155^{*}$ $\sigma = 0,01529^{**}$
2	95,9	2,28	1,14	$f = 0$ $\overline{f} = 0,00107^{*}$ $\sigma = 0,01066^{**}$
4	43,6	5,01	1,25	$f = 0$ $\overline{f} = 0,00132^{*}$ $\sigma = 0,01327^{**}$
8	20,52	10,66	1,33	$f = 0$ $\overline{f} = 0,00117^{*}$ $\sigma = 0,01164^{**}$
16	11,41	19,16	1,2	$f = 0$ $\overline{f} = 0,00118^{*}$ $\sigma = 0,01262^{**}$
32	8,72	25,08	0,78	$f = 0$ $\overline{f} = 0,00095^{*}$ $\sigma = 0,00978^{**}$

* Valores médios e ** desvios padrões.

onde $z_i = 1 + \frac{x_i - 1}{4}$ e $-10 \leq x_i \leq 10, \;\; i = 1, \cdots, n$. O problema foi resolvido para o caso n=30. Sabe-se que $x_{\text{ótimo}} = (1, \cdots, 1)$ e $f(x_{\text{ótimo}}) = 0$. A Tab. 5.7 resume os resultados encontrados para $f(x_{\text{ótimo}})$.

Novamente, a redução do tempo de execução do algoritmo em paralelo é muito significativa. Para a função de Levy o tempo de processamento sequencial é 25 vezes maior que o tempo necessário para 32 processadores resolver o problema. A solução ótima é encontrada em todos os casos e os devios padrões dos valores ótimos da função objetivo são muito baixos.

Pode-se observar na Tab. 5.7 que com os testes com a função de Levy apenas a simulação com 32 processadores não apresentou valores super lineares para o speedup e a eficiência.

5.2.4 Função de Shubert

A função de Shubert é uma função de duas variáveis e por ser multimodal, possuindo 18 mínimos globais, torna-se uma excelente opção para testar algoritmos de

Tabela 5.8 – Função teste de Shubert.

Número de processadores	Tempo em segundos	Speedup	Eficiência	$f(x_{\text{ótimo}})$
1	66,85	1	1	$f = -186,7309$ $\overline{f} = -186,7309^{*}$ $\sigma = 0,00002^{**}$
2	32,15	2,08	1,04	$f = -186,7309$ $\overline{f} = -186,7309^{*}$ $\sigma = 0,00004^{**}$
4	16,91	3,95	0,98	$f = -186.7309$ $\overline{f} = -186,7308^{*}$ $\sigma = 0,00027^{**}$
8	8,9	7,51	0,93	$f = -186.7309$ $\overline{f} = -186,7308^{*}$ $\sigma = 0,00024^{**}$
16	4,23	15,8	0,99	$f = -186.7309$ $\overline{f} = -186,7308^{*}$ $\sigma = 0,0009^{**}$
32	3,01	22,21	0,69	$f = -186.7309$ $\overline{f} = -186,7306^{*}$ $\sigma = 0,0035^{**}$

* Valores médios e ** desvios padrões.

otimização. Seu gráfico está ilustrado na Fig. 5.3 (a) e os valores ótimos podem ser vistos na Tab. 5.8. A função de Shubert é escrita como:

$$min\ f(x) = \left(\sum_{i=1}^{5} icos((i+1)x_1 + i)\right) \cdot \left(\sum_{i=1}^{5} icos((i+1)x_2 + i)\right) \tag{5.26}$$

Tem-se que $f(x_{\text{ótimo}}) = -186,7309$. Observe que em todos os casos a solução ótima foi encontrada e que os valores ótimos do desvio padrão das funções objetivos foram muito baixos, indicando assim que mesmo dividindo a execução do problema em vários processadores a solução ótima foi obtida sem muita variância no seu valor.

Note que para as simulações com a função de Shubert apenas o caso com 2 processadores apresentou valores super lineares para o speedup e a eficiência.

Tabela 5.9 – Função teste de Rastrigin.

Número de processadores	Tempo em segundos	Speedup	Eficiência	$x_{\text{ótimo}}$	$f(x_{\text{ótimo}})$
1	54,73	1	1	$(0;0)$ $(0,1219 \cdot 10^{-7}; 0,6467 \cdot 10^{-7})^{*}$ $(0,1208 \cdot 10^{-6}; 0,6504 \cdot 10^{-6})^{**}$	$f = 0$ $\overline{f} = 0,8 \cdot 10^{-10^{*}}$ $\sigma = 0,8 \cdot 10^{-9^{**}}$
2	25,71	2,13	1,06	$(0;0)$ $(0,6589 \cdot 10^{-7}; 0,2375 \cdot 10^{-7})^{*}$ $(0,6884 \cdot 10^{-6}; 0,3241 \cdot 10^{-6})^{**}$	$f = 0$ $\overline{f} = 0,11 \cdot 10^{-9^{*}}$ $\sigma = 0,11 \cdot 10^{-8^{**}}$
4	13,09	4,18	1,04	$(0;0)$ $(0,4967 \cdot 10^{-7}; 0,287 \cdot 10^{-8})^{*}$ $(0,1144 \cdot 10^{-5}; 0,931 \cdot 10^{-6})^{**}$	$f = 0$ $\overline{f} = 0,43 \cdot 10^{-9^{*}}$ $\sigma = 0,5 \cdot 10^{-8^{**}}$
8	7,36	7,44	0,93	$(0;0)$ $(0,5623 \cdot 10^{-7}; -0,837 \cdot 10^{-8})^{*}$ $(0,1371 \cdot 10^{-5}; 0,758 \cdot 10^{-6})^{**}$	$f = 0$ $\overline{f} = 0,48 \cdot 10^{-9^{*}}$ $\sigma = 0,8 \cdot 10^{-8^{**}}$
16	3,23	16,94	1,05	$(0;0)$ $(0,2026 \cdot 10^{-7}; 0,2749 \cdot 10^{-7})^{*}$ $(0,1676 \cdot 10^{-5}; 0,1694 \cdot 10^{-5})^{**}$	$f = 0$ $\overline{f} = 0,112 \cdot 10^{-8^{*}}$ $\sigma = 0,16 \cdot 10^{-7^{**}}$
32	2,52	21,72	0,68	$(0;0)$ $(-0,1176 \cdot 10^{-7}; 0,3829 \cdot 10^{-7})^{*}$ $(0,7308 \cdot 10^{-5}; 0,4985 \cdot 10^{-5})^{**}$	$f = 0$ $\overline{f} = 0,1552 \cdot 10^{-7^{*}}$ $\sigma = 0,355 \cdot 10^{-6^{**}}$

* Valores médios e ** desvios padrões.

5.2.5 Função de Rastrigin

Seja o mínimo da função de Rastrigin representado pela seguinte equação:

$$min\ f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10cos(2\pi x_i)\right] \tag{5.27}$$

sendo que $-5,12 \leq x_i \leq 5,12, \quad i = 1, \cdots, n$. O problema foi resolvido para o caso n=2 onde $x_{\text{ótimo}} = (0;0)$ e $f(x_{\text{ótimo}}) = 0$. A Tab. 5.9 apresenta os resultados ótimos e sua representação gráfica pode ser observada na Fig. 5.3 (b).

A solução ótima, $x_{\text{ótimo}}$, foi encontrada com no mínimo 5 casas decimais de precisão. O tempo de execução do programa para 32 processadores foi menor que 3 segundos e o desvio padrão dos valores ótimos da função objetivo foram muito baixos, comprovando assim a eficiência do algoritmo em paralelo.

Assim como ocorreu nos testes com a função de Beale, as simulações com 2, 4 e 16 processadores da função de Rastrigin apresentaram speedup e eficiência com valores super lineares, conforme exposto na Tab. 5.9.

5.2.6 Problemas Restritos

Foram analizados dois problemas testes com restrições. Normalmente, os métodos heurísticos são desenvolvidos para problemas irrestritos. Mas no desenvolvimento

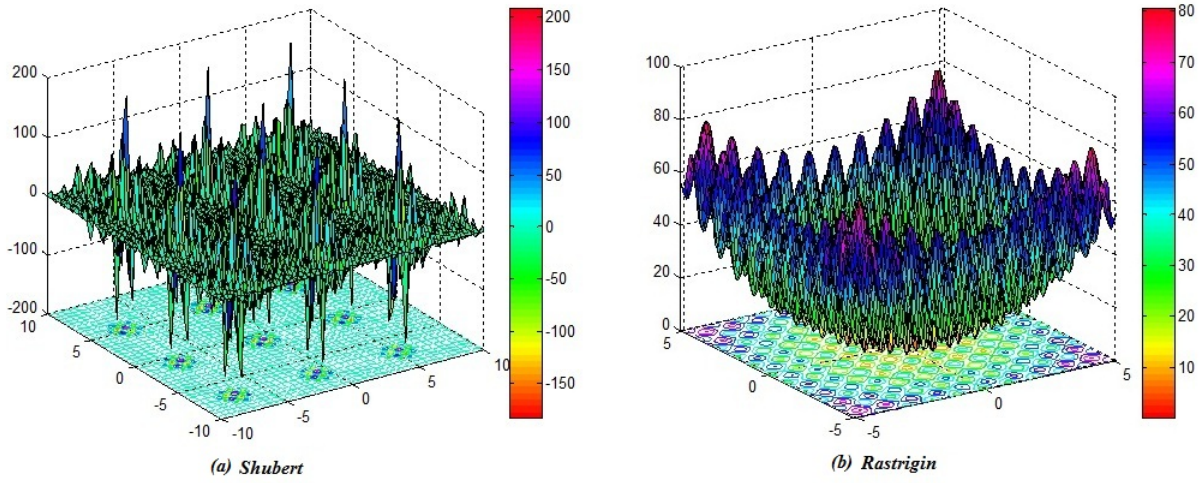


Figura 5.3 – Representação gráfica das funções: (a) Shubert e (b) Rastrigin.

do código computacional para o EDMP foram consideradas a presença de restrições utilizando o Método da Penalidade. Para estes problemas, adotou-se um fator de penalidade elevado da ordem 10^5, conforme Eqs. (2.10) e (2.11).

Problema Restrito 1: O primeiro problema restrito testado, dado pela Eq. (5.28), envolve treze variáveis de projeto e nove restrições representadas pelas Eqs. de (5.29) a (5.37).

$$min\ f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i \tag{5.28}$$

Sujeito a:

$$g_1(x) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \tag{5.29}$$

$$g_2(x) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \tag{5.30}$$

$$g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \tag{5.31}$$

$$g_4(x) = -8x_1 + x_{10} \leq 0 \tag{5.32}$$

$$g_5(x) = -8x_2 + x_{11} \leq 0 \tag{5.33}$$

$$g_6(x) = -8x_3 + x_{12} \leq 0 \tag{5.34}$$

$$g_7(x) = -2x_4 - x_5 + x_{10} \leq 0 \tag{5.35}$$

Tabela 5.10 – Problema restrito 1.

Número de processadores	Tempo em segundos	Speedup	Eficiência	$f(x_{\text{ótimo}})$
1	91,74	1	1	$f = -15$ $\overline{f} = -14,0906^{*}$ $\sigma = 1,1327^{**}$
2	45,16	2,03	1,02	$f = -15$ $\overline{f} = -14,3453^{*}$ $\sigma = 1,0125^{**}$
4	21,58	4,25	1,06	$f = -15$ $\overline{f} = -14,5726^{*}$ $\sigma = 0,8623^{**}$
8	10,68	8,59	1,07	$f = -15$ $\overline{f} = -14,8863^{*}$ $\sigma = 0,4995^{**}$
16	4,98	18,42	1,15	$f = -15$ $\overline{f} = -14,0123^{*}$ $\sigma = 1,2119^{**}$
32	4,03	22,76	0,71	$f = -15$ $\overline{f} = -14,113^{*}$ $\sigma = 1,2325^{**}$

* Valores médios e ** desvios padrões.

$$g_8(x) = -2x_6 - x_7 + x_{11} \leq 0 \tag{5.36}$$

$$g_9(x) = -2x_8 - x_9 + x_{12} \leq 0 \tag{5.37}$$

O resultado ótimo para o problema restrito 1 é $x_{\text{ótimo}} = (1, 1, \cdots, 1, 3, 3, 3, 1)$ e $f(x_{\text{ótimo}}) = -15$. Os limites laterais das variáveis são:

$$0 \leq x_i \leq 1, \;\; i = 1, \cdots, 9, 13, \;\; \text{e} \;\; 0 \leq x_i < 100, \;\; i = 10, 11, 12$$

Na Tab. 5.10 são apresentados um resumo dos valores encontrados. Em todos os casos foi obtido $x_{\text{ótimo}} = (1; 1; 1; 1; 1; 1; 1; 1; 1; 3; 3; 3; 1)$, para simplificar foi omitido os valores médios e desvios padrões de $x_{\text{ótimo}}$ na tabela. Deve-se ressaltar que para estes resultados todas as restrições foram obedecidas.

Além disso, pode-se observar na Tab. 5.10 que apenas a simulação com 32 processadores não apresentou valores super lineares para o speedup e a eficiência.

Problema Restrito 2: O segundo problema teste, dado pela Eq. (5.38), con-

sidera sete variáveis, cujos limites laterais são $-10 \leq x_i \leq 10, \quad i = 1, \cdots, 7$ e quatro restrições representadas pelas Eqs. de (5.39) a (5.42).

$$min\ f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \tag{5.38}$$

Sujeito a:

$$g_1(x) = 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 - 127 \leq 0 \tag{5.39}$$

$$g_2(x) = 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 - 282 \leq 0 \tag{5.40}$$

$$g_3(x) = 23x_1 + x_2^2 + 6x_6^2 - 8x_7 - 196 \leq 0 \tag{5.41}$$

$$g_4(x) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0 \tag{5.42}$$

Os pontos ótimos calculados podem ser observados na Tab. 5.11. De forma similar ao exemplo anterior todas as restrições foram obedecidas e, independente do número de processadores utilizados, os valores ótimos encontrados para este problema foi

$$x_{\text{ótimo}} = (2,330499; 1,951372; -0,4775414; 4,365726; -0,6244870; 1,038131; 1,594227).$$

Novamente, pode-se observar na Tab. 5.11 que apenas a simulação com 32 processadores não apresentou valores super lineares para o speedup e a eficiência.

Os excelentes resultados encontrados nos problemas resolvidos, restritos e irrestritos, utilizando Evolução Diferencial Melhorada em Paralelo permite validar o algoritmo, pois o mesmo mostrou-se preciso e robusto.

Os valores obtidos com as métricas speedup e eficiência indicam que toda a capacidade computacional disponível no sistema foi utilizada para ganhar velocidade na execução do algoritmo em paralelo visto que em todos os problemas resolvidos com 2, 4, 8 e 16 processadores obteve-se ou eficiência super linear ou quase linear, pois o valor

Tabela 5.11 – Problema restrito 2.

Número de processadores	Tempo em segundos	Speedup	Eficiência	$f(x_{\text{ótimo}})$
1	70,72	1	1	$f = 680,6300573$ $\overline{f} = 680,6316054^{*}$ $\sigma = 0,0118828^{**}$
2	32,88	2,15	1,07	$f = 680,6300573$ $\overline{f} = 680,6313112^{*}$ $\sigma = 0,0090316^{**}$
4	16,99	4,16	1,04	$f = 680,6300573$ $\overline{f} = 680,6319933^{*}$ $\sigma = 0,0146679^{**}$
8	8,54	8,28	1,03	$f = 680,6300573$ $\overline{f} = 680,6326813^{*}$ $\sigma = 0,0199507^{**}$
16	3,98	17,77	1,11	$f = 680,6300573$ $\overline{f} = 680,6333892^{*}$ $\sigma = 0,0258041^{**}$
32	3,18	22,23	0,69	$f = 680,6300573$ $\overline{f} = 680,6340817^{*}$ $\sigma = 0,0330548^{**}$

* Valores médios e ** desvios padrões.

da eficiência nestes casos ou foi maior que 1 ou muito próximo à 1. No entanto, ao usar 32 processadores para resolver os problemas apresentados a eficiência diminui ficando em torno de 70%, isto é resultante do aumento da comunicação entre os processadores.

A eficiência super-linear do código implementado em paralelo ocorre principalmente devido ao fato de que o algoritmo desenvolvido converge mais rapidamente quando divide a população em subpopulações. Além disso, também há independência entre os processos, independência esta que faz com que o tempo gasto entre as comunicações seja muito pequena.

Estes excelentes resultados obtidos com estes primeiros testes indicam que o algoritmo da EDMP está altamente paralelizável e que a comunicação entre os processadores é mínima. Portanto, pode-se garantir grande diminuição do tempo de execução do programa em relação ao algoritmo sequencial.

Nas próximas seções serão apresentadas simulações de problemas práticos da engenharia mecânica cujos resultados também confirmarão o quanto o algoritmo da EDMP é rápido nas suas execuções.

5.3 Otimização de Sistemas Robóticos

O objetivo desta seção é avaliar o algoritmo Evolução Diferencial Melhorada implementada em paralelo na resolução de um problema prático da engenharia mecânica que consiste em obter o projeto ideal de um robô manipulador levando em conta as características do seu espaço de trabalho, conforme apresentado por OLIVEIRA (2011) e SARAMAGO; BERGAMASCHI; OLIVEIRA (2007). Para esta finalidade, um problema multiobjetivo de otimização é formulado para se obter os parâmetros ideais para o robô. Os resultados mostram que o procedimento representa uma alternativa promissora para este tipo de problema.

Será considerado os robôs manipuladores com três juntas rotacionais, cuja abreviação é dada por 3R, conforme Fig. 5.4.

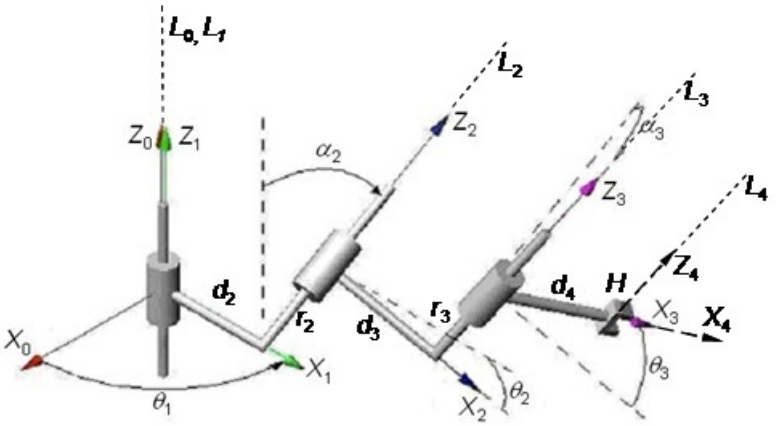


Figura 5.4 – Esquema cinemático de um robô manipulador 3R com seus parâmetros de projeto. Fonte: SARAMAGO; BERGAMASCHI; OLIVEIRA (2007).

O mecanismo de um manipulador em série é um conjunto cinemático constituído de uma sucessão de corpos rígidos ligados entre si por juntas rotacionais e/ou prismáticas. Uma forma de representação cinemática comum é a utilização dos parâmetros de Denavit-Hartenberg (DH). Dois sistemas de referência R_{j-1} e R_j são relacionados pela matriz de transformação T_j^{j-1}, dada por:

$$T_j^{j-1} = \begin{pmatrix} c_j & -s_j & 0 & d_j \\ c\alpha_j s_j & c\alpha_j c_j & -s\alpha_j & -r_j s\alpha_j \\ s\alpha_j s_j & s\alpha_j c_j & c\alpha_j & r_j c\alpha_j \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

sendo $c_j = cos(\theta_j), s_j = sen(\theta_j), c\alpha_j = cos(\alpha_j)$ e $s\alpha_j = sen(\alpha_j), j = 1, ..., n+1$.

Assim, ao efetuador de um robô de n graus de liberdade é associado um sistema de coordenadas, T_n^0, que é uma matriz de transformação homogênea $T_n^0 = T_1^0 \cdot T_2^1 \cdots T_j^{j-1} T_{j+1}^j \cdots T_n^{n-1}$.

Trabalhando com estas matrizes, obtém-se o Modelo Geométrico Direto (MGD) de um manipulador 3R, dado por:

$$\begin{aligned} x = \; & [d_2 + (r_3 s\alpha_3 - d_4 c\alpha_3 s_3)\, s_2 + (d_3 + d_4 c_3)\, c_2]\, c_1 + \{s\alpha_2\, (r_2 + r_3 c\alpha_3 + d_4 s\alpha_3 s_3) \\ & + c\alpha_2\, [(r_3 s\alpha_3 - d_4 c\alpha_3 s_3)\, c_2 - (d_3 + d_4 c_3)\, s_2]\}\, s_1 \\ y = \; & [d_2 + (r_3 s\alpha_3 - d_4 c\alpha_3 s_3)\, s_2 + (d_3 + d_4 c_3)\, c_2]\, s_1 - \{s\alpha_2\, (r_2 + r_3 c\alpha_3 + d_4 s\alpha_3 s_3) \\ & + c\alpha_2\, [(r_3 s\alpha_3 - d_4 c\alpha_3 s_3)\, c_2 - (d_3 + d_4 c_3)\, s_2]\}\, c_1 \\ z = \; & c\alpha_2\, (r_2 + r_3 c\alpha_3 + d_4 s\alpha_3 s_3) - s\alpha_2\, [(r_3 s\alpha_3 - d_4 c\alpha_3 s_3)\, c_2 - (d_3 + d_4 c_3)\, s_2] \end{aligned} \tag{5.43}$$

Com as Eqs. (5.43) pode-se obter o determinante da matriz Jacobiana J. Será utilizada nesta pesquisa a matriz Jacobiana de Base obtida por OLIVEIRA (2011) e BAILI (2004) que é dada por:

$$J = \begin{pmatrix} -s_3 c_2 d_4 - c_2 r_2 & r_3 & -s_3 d_4 \\ s_3 s_2 d_4 + s_2 r_2 & d_3 + c_3 d_4 & 0 \\ c_2 d_3 + c_2 c_3 d_4 + d_2 + s_2 r_3 & 0 & c_3 d_4 \end{pmatrix} \tag{5.44}$$

cujo determinante é dado por:

$$det(J) = d_4 \left\{(d_3 + d_4 c_3)[d_2 s_3 + (d_3 s_3 - r_2 c_3) c_2] + r_3 s_2 (d_3 s_3 - r_2 c_3)\right\}. \tag{5.45}$$

Um problema de otimização multiobjetivo será formulado para obtenção dos parâmetros ideais do robô manipulador. A formulação matemática para calcular o volume do espaço de trabalho, a rigidez do sistema e a destreza do robô será dada em seguida.

Formulação do Problema de Otimização

O trabalho proposto consiste na síntese dimensional de um manipulador 3R ortogonal (α_2= -90º e α_3 = 90º), onde o projeto ótimo corresponde àquele que otimiza algumas características desejáveis para a melhor performance do robô à executar tarefas. O objetivo é a obtenção de um projeto ótimo que considera a maximização do volume de trabalho do manipulador (V), a maximização da rigidez do mecanismo (R) e a otimização de sua destreza por meio da minimização do índice de isotropia (D). Assim, o problema de otimização pode ser formulado como:

$$\text{Maximizar } F(X) = [V \ -D \ R] \tag{5.46}$$

onde

$$X = (1, d_3, d_4, r_2, r_3)^T, \qquad 0,1 \leq d_3, d_4, r_2, r_3 \leq 3,0 \tag{5.47}$$

sendo que as variáveis d_3, d_4, r_2 e r_3 estão representadas na Fig. 5.4.

O trabalho atual utiliza a modelagem desenvolvida por OLIVEIRA et al (2008) e OLIVEIRA e SARAMAGO (2010), que aplicou em sua pesquisa Algoritmo Genético e Evolução Diferencial para solucionar o problema de otimização.

Volume do Espaço de Trabalho

O volume do espaço de trabalho, V, é o volume do sólido de revolução obtido pela rotação da seção radial em torno do eixo z. Assim, usando o Teorema de Pappus-

Guldin, conforme apresentado na Fig. 5.5, o volume é dado através da equação:

$$V = 2\pi r_g A, \tag{5.48}$$

sendo A a área da seção radial plana que é coberta pela família de curvas e r_g a coordenada do baricentro.

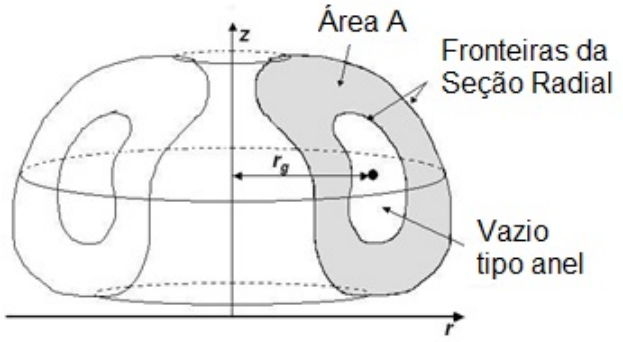


Figura 5.5 – Cálculo do volume do espaço de trabalho de manipuladores 3R. Fonte: OLIVEIRA (2011).

Esta pesquisa propõe uma formulação numérica para aproximar o cálculo da área da seção radial através de sua discretização em uma malha retangular (SARAMAGO; BERGAMASCHI; OLIVEIRA (2007)). Inicialmente, devem-se obter os valores extremos dos vetores r e z, ou seja,

$$\begin{aligned} r_{min} &= min\{r\} \quad e \quad r_{max} = max\{r\} \\ z_{min} &= min\{z\} \quad e \quad z_{max} = max\{z\} \end{aligned} \tag{5.49}$$

Adotando-se o número de subintervalos desejados para a discretização ao longo de r e z (n_r e n_z), pode-se calcular as dimensões das áreas elementares da malha através das seguintes expressões:

$$\Delta_r = \frac{r_{max} - r_{min}}{n_r} \quad e \quad \Delta_z = \frac{z_{max} - z_{min}}{n_z} \tag{5.50}$$

Utilizando as Eqs. (5.43), do modelo geométrico direto do manipulador, calcula-se todos os pontos da família de curvas que compõem a seção radial do espaço de trabalho. Dado um determinado ponto (r, z), determina-se sua posição dentro da malha

de discretização, através do seguinte controle de índices:

$$i = int\left(\frac{r - r_{min}}{\Delta_r}\right) + 1 \quad e \quad j = int\left(\frac{z - z_{min}}{\Delta_z}\right) + 1 \tag{5.51}$$

Conforme mostrado no esquema da Fig. 5.6, o ponto da malha que pertence ao espaço de trabalho será identificado como $P_{ij} = 1$, caso contrário terá valor nulo, ou seja:

$$P_{ij} = \begin{cases} 0, & se\ P_{ij} \notin W(H) \\ 1, & se\ P_{ij} \in W(H) \end{cases} \tag{5.52}$$

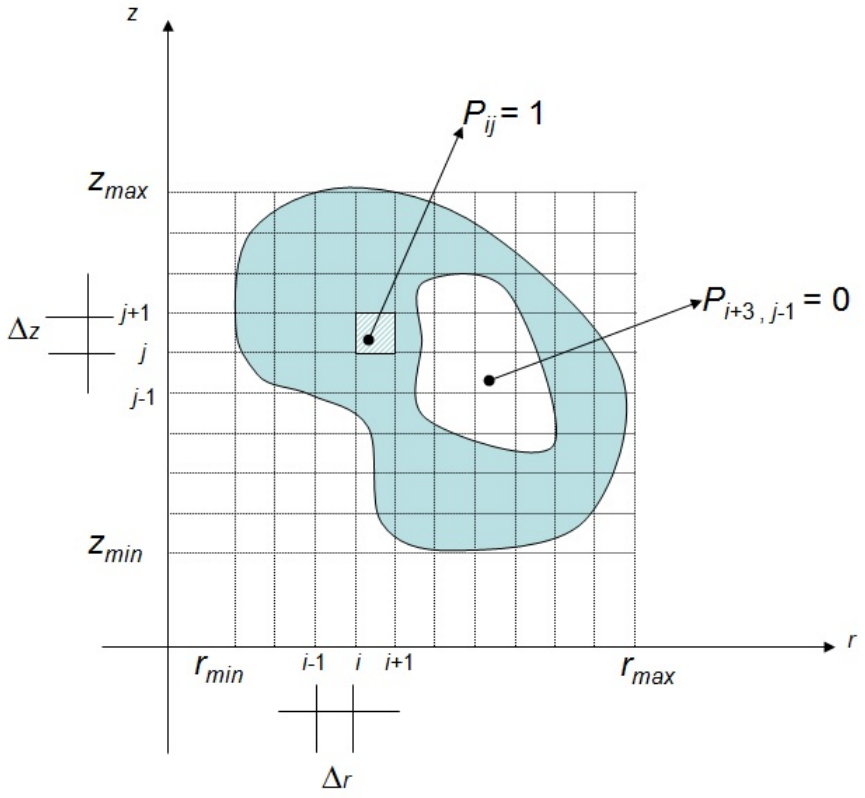


Figura 5.6 – Discretização da seção radial usando malha retangular. Fonte OLIVEIRA et al (2006).

Desta forma, a área total é obtida pela soma de todas as áreas elementares da malha que estão contidas, totalmente ou parcialmente, na seção radial, conforme Eq. (5.53). Observa-se que apenas os pontos pertencentes ao espaço de trabalho

contribuem para o cálculo da área:

$$A = \sum_{i=1}^{i_{max}} \sum_{j=1}^{j_{max}} (P_{ij} \Delta_r \Delta_z) \tag{5.53}$$

A coordenada do baricentro é calculada considerando a soma dos baricentros de cada área elementar, dividida pela área total, dada por:

$$r_g = \frac{\sum_{i=1}^{i_{max}} \sum_{j=1}^{j_{max}} (P_{ij} \Delta_r \Delta_z) \left((i-1) \Delta_r + \frac{\Delta_r}{2} + r_{min} \right)}{A} \tag{5.54}$$

Finalmente, conhecendo-se os valores da área e do baricentro da seção radial, dados pelas Eqs. (5.53) e (5.54), pode-se calcular o volume do espaço de trabalho do manipulador usando a Eq. (5.48). Desta forma, a Eq. (5.48) representa a função objetivo a ser maximizada para obter o maior volume do espaço de trabalho possível.

Rigidez do Mecanismo Serial

Pode-se definir rigidez como sendo a capacidade de um sistema mecânico de suportar cargas sem grandes mudanças em sua geometria. Assim, a rigidez é uma característica mecânica que descreve o comportamento de uma estrutura sujeita às forças estáticas em termos da deflexão elástica. Desta forma, o estudo da rigidez de um sistema equivale a obter a matriz de rigidez, K, da estrutura analisada, que representa a medida da capacidade da estrutura de resistir às deformações devido à ação de esforços externos.

A fim de se obter o modelo da rigidez da estrutura pode-se adotar a matriz de análise do mecanismo que lida com a estrutura como uma combinação de elementos e nós conforme proposto por DEBLAISE; HERNOT; MAURINE (2006) ou ainda com métodos baseados no cálculo da matriz Jacobiana do mecanismo serial de acordo com EL-KHASAWNEH e FERREIRA (1999) e COMPANY; PIERROT; FAUROUX (2005).

Neste trabalho, o modelo de rigidez da estrutura é obtido a partir da matriz Jacobiana do mecanismo serial, considerando os elementos como sendo molas. A matriz

de rigidez do sistema de juntas do mecanismo no espaço cartesiano é dada por:

$$K = [J]^T K_j [J] \tag{5.55}$$

onde K_j é a matriz diagonal $n \times n$. Para o caso estudado, mecanismo serial com 3 juntas rotacionais, tem-se $K_j = diag[k_1, k_2, k_3]$ e J a matriz Jacobiana, dada na Eq. (5.44).

Além disso, os elementos da diagonal da matriz de rigidez são utilizados como os valores da rigidez do sistema. Estes elementos representam a rigidez pura em cada direção, e refletem a rigidez/inflexibilidade das ferramentas da máquina de forma mais clara e direta. Assim, a função objetivo para otimizar a rigidez do sistema pode ser escrita conforme a Eq. (5.56). Neste caso, a rigidez R deve ser maximizada:

$$R = K_{11} + K_{22} + K_{33} \tag{5.56}$$

onde $K_{ii}, i = 1, 2, 3$, representa os elementos da diagonal da matriz de rigidez do mecanismo.

Destreza

O índice de desempenho de um sistema mecânico robótico é uma quantidade escalar que mede o quão satisfatório o sistema se comporta com relação à transmissão de movimento e força. Este índice pode ser definido para todos os tipos de sistemas mecânicos robóticos, particularmente, os manipuladores seriais. Existem vários índices de desempenho definidos na literatura que podem ser vistos em PAUL e STEVENSON (1983), VINOGRADOV et al (1971), YANG e LAI (1985) e YOSHIKAWA (1985).

Neste trabalho, será adotado o índice de isotropia, D, que é definido como o índice de condicionamento da matriz Jacobiana J. Tal índice pode ser escrito como a relação entre o maior e o menor valor singular de J:

$$D = \frac{|\lambda_{max}(J)|}{|\lambda_{min}(J)|} \tag{5.57}$$

sendo que λ_{max} e λ_{min} significam, respectivamente, os valores singulares máximo e mínimo de J.

Na robótica, o índice de isotropia reflete a precisão resultante das velocidades operacionais de entrada a partir das velocidades articulares calculadas usando a inversa da matriz J. Este índice pode alcançar valores no intervalo [1; ∞].

O valor 1 significa uma isotropia: o elipsóide das velocidades toma a forma de uma esfera. Fisicamente, quando o manipulador está em uma posição isotrópica, o efetuador terá a mesma facilidade para se deslocar em todas as direções. Assim, manipuladores isotrópicos são aqueles cuja Jacobiana pode alcançar valores isotrópicos, neste caso, $D = 1$. Quanto mais próximo de 1, maior a agilidade para executar tarefas no espaço de trabalho.

O índice de isotropia pode indicar distorção no espaço das variáveis. Quanto maior for esta distorção, maior será o índice de isotropia. Portanto, para a otimização da destreza, o índice de isotropia deve ser minimizado.

5.3.1 Simulação Numérica do Problema Irrestrito

Como mencionado anteriomente, o objetivo do esquema proposto para o projeto do manipulador é a otimização da dimensão do robô ortogonal 3R considerando o espaço de trabalho (V), a rigidez do mecanismo (R) e a destreza do manipulador (D), conforme mostrado nas Eqs. (5.46) e (5.47).

O volume do espaço de trabalho é dado pela Eq. (5.48) e a rigidez do mecanismo pela Eq. (5.56). A fim de otimizar a destreza, o índice de isotropia, dado pela Eq. (5.57), foi minimizado.

No intuito de avaliar a performance entre as técnicas ECE, ED e EDMP os seguintes pontos devem ser salientados:

- parâmetros da ED: tamanho da população (9), taxa de pertubação (0,8), probabilidade de cruzamento (0,6);
- parâmetros da EDMP: tamanho da população 36 indivíduos divididos entre 4 processadores, taxa de pertubação (0,8), probabilidade de cruzamento (0,6);

- parâmetros da ECE: tamanho da população (9 indivíduos em cada complexo), o ponto inicial é o ponto médio dos limites laterais das variáveis de projeto, a saber $x_0 = [1,45 \;\; 1,45 \;\; 1,45 \;\; 1,45]$;

- critério de parada usado em ED foi número máximo de iterações ou estagnação do valor da função objetivo. O critério de parada usado em ECE foi a série geométrica normalizada dos parâmetros menor do que 0,0001. O critério de parada usado em EDMP foi o número máximo de iterações;

- ED e EDMP foi executado 20 vezes para obter a média dos valores apresentados nas tabelas;

- parâmetros do robô considerados: $d_2 = 1$, $d_3 = x(1)$, $d_4 = x(2)$, $r_2 = x(3)$, $r_3 = x(4)$, $a_2 = -\pi/2$, $a_3 = \pi/2$, o tamanho do passo para calcular a Jacobiana: 0,03 e malha: 50$\times$50;

- todas as simulações foram resolvidas usando um processador Intel® Core^{TM} i5-430M.

Nas seguintes Tabs., 5.12 e 5.13, estão apresentados os valores ótimos encontrados usando os métodos de otimização Evolução Diferencial, Evolução com Conjuntos Embaralhados (com 2, 4 e 8 conjuntos) e Evolução Diferencial Melhorada implementada em paralelo. É importante mencionar que parte dos resultados expostos nas Tabs. 5.12 e 5.13 foram apresentados por BRANDÃO et al (2012) no World Congress on Computational Mechanics.

Tabela 5.12 – Valores ótimos considerando o método do Critério Global (Métrica L_{2R}).

Algoritmo	Volume [u.v.]	Destreza	Rigidez [u.r.]	$[d_3[u.l.] \;\; d_4[u.l.] \;\; r_2[u.l.] \;\; r_3[u.l.]]$	Tempo	N_{ava}
ED	1902,70 1903,30* 0,74**	1,04 1,04* 0,01**	105,68 105,68* 0,01**	[3 3 3 0,1] [3,00 3,00 3,00 0,11]* [0 0 0 0,01]**	8,03h	150
ECE 2 conjuntos	1905,10 1844,85* 132.71**	1,14 1,14* 0.12**	105,90 103,51* 5.19**	[3,00 3,00 3,00 0,37] [2,95 2,97 2,97 0,39]* [0.15 0.07 0.09 0.39]**	3,05h	1462
ECE 4 conjuntos	1908,20 1805,18* 137,30**	1,11 1,13* 0,13**	105,82 102,17* 5,13**	[3,00 3,00 3,00 0,30] [2,95 2,94 2,90 0,34]* [0,11 0,12 0,14 0,34]**	7,24h	3468
ECE 8 conjuntos	1903,40 1832,20* 142,31**	1,08 1,10* 0,06**	105,71 103,23* 5,15**	[3,00 3,00 3,00 0,23] [2,98 2,96 2,91 0,26]* [0,06 0,12 0,21 0,30]**	14,23h	6800
EDMP executado em 4 processadores	1902,76 1902,76* 4e-13**	1,03 1,03* 6e-16**	105,67 105,67* 7e-14**	[3,00 3,00 3,00 0,1] [3,00 3,00 3,00 0,1]* [0,0 0,0 0,0 4e-17]**	45s	330

* Média dos valores, ** desvio padrão e N_{ava} número de avaliações da função objetivo.

Tabela 5.13 – Valores ótimos considerando o método da Ponderação dos Objetivos.

Coef.	Algoritmo	Volume [u.v.]	Destreza	Rigidez [u.r.]	$[d_3[u.c.]\ \ d_4[u.c.]\ \ r_2[u.c.]\ \ r_3[u.c.]]$	Tempo	N_{ava}
$w_1 = 0.05$ $w_2 = 0.90$ $w_3 = 0.05$	ED	1902,70 1703,20* 490,67**	1,04 1,04* 0,01**	105,68 97,11* 21,19**	[3,00 3,00 3,00 0,10] [2,87 2,82 2,80 0,10]* [0,61 0,48 0,65 0]**	15,45h	180
	ECE 2 conjuntos	1892,70 1164,10* 722,93**	1,04 1,07* 0,05**	105,42 72,17* 34,99**	[3,00 3,00 3,00 0,11] [1,93 2,47 2,70 0,19]* [1,23 0,60 0,50 0,15]**	4,78h	2278
	ECE 4 conjuntos	1896,30 1621,40* 374,60**	1,04 1,07* 0,05**	105,49 94,31* 16,70**	[3,00 3,00 3,00 0,11] [2,80 2,82 2,80 0,17]* [0,44 0,31 0,32 0,15]**	6,09h	2856
	ECE 8 conjuntos	1879,00 1635,90* 381,17**	1,04 1,06* 0,03**	104,88 94,68* 16,79**	[3,00 2,99 2,99 0,11] [2,81 2,82 2,84 0,13]* [0,33 0,31 0,30 0,05]**	11,11h	5304
	EDMP executado em 4 processadores	1902,75 1030,33* 873,13**	1,03 1,04* 0,02**	105,67 62,64* 43,13**	[3,00 3,00 3,00 0,1] [2,05 2,02 2,02 0,52]* [0,98 0,97 0,98 0,42]**	48s	360
$w_1 = 1/3$ $w_2 = 1/3$ $w_3 = 1/3$	ED	1902,70 1934,90* 99,03**	1,04 1,09* 0,18**	105,68 106,93* 3,87**	[3,00 3,00 3,00 0,10] [3,00 3,00 3,00 0,34]* [0 0 0 0,72]**	28,75h	900
	ECE 2 conjuntos	1902,60 1799,20* 143,27**	1,04 1,14* 0,20**	105,66 102,02* 5,26**	[3,00 3,00 3,00 0,10] [2,94 2,94 2,88 0,38]* [0,14 0,12 0,18 0,56]**	2,71h	1258
	ECE 4 conjuntos	1902,60 1863,30* 81,37**	1,04 1,13* 0,22**	105,67 104,45* 3,20**	[3,00 3,00 3,00 0,11] [2,94 2,98 2,94 0,39]* [0,12 0,05 0,12 0,74]**	6,81h	3128
	ECE 8 conjuntos	1902,60 1839,90* 105,97**	1,04 1,09* 0,12**	105,67 103,39* 3,91**	[3,00 3,00 3,00 0,11] [2,95 2,98 2,95 0,24]* [0,12 0,05 0,11 0,31]**	11,53h	5576
	EDMP executado em 4 processadores	2291,61 2289,68* 2,88**	1,71 1,71* 0,004**	120,75 120,72* 0,08**	[3,00 3,00 3,00 2,64] [3,00 3,00 3,00 2,64]* [0,0 0,0 0,0 0,007]**	43s	320

* Média dos valores, ** desvio padrão e N_{ava} número de avaliações da função objetivo.

Vale salientar que os resultados ótimos apresentados na Tab. 5.13 são dependentes dos valores dos coeficientes de ponderação. Esta tabela mostra o caso em que a destreza é priorizada (w_2=0,9) e o caso em que todas as prioridades são iguais ($w_1 = w_2 = w_3$=1/3).

Considerando que os valores ideais do volume do espaço de trabalho, da destreza e da rigidez são, V_{ideal}=2382.7412 [u.v.], D_{ideal}=1.0256 e R_{ideal}=125.0769 [u.r.], respectivamente, pode-se concluir que os métodos são eficazes em resolver os problemas visto que os resultados obtidos aproximam dos valores ideais. Comparando os resultados encontrados com ECE, ED e EDMP é possível observar que os valores são muito similares. No entanto, para o problema estudado, o método EDMP mostrou-se muito mais rápido. Por exemplo, para encontrar a solução ótima do problema proposto considerando a métrica L_{2R} (Tab. 5.12) o menor tempo de execução entre os algoritmos sequenciais foi de 3,05 horas enquanto EDMP precisou de apenas 45 segundos. Este comportamento também se repetiu considerando o método dos objetivos ponderados.

Outros problemas de otimização multiobjetivo de sistemas robóticos irrestri-

tos foram simulados com EDMP. Por exemplo, em BRANDÃO; SARAMAGO; DORICIO (2013) é apresentado e comparado as soluções obtidas com ED e EDMP para encontrar a solução ótima de sistemas robóticos considerando como variáveis de projeto os seguintes parâmetros geométricos do robô: $d_3, d_4, r_2, r_3, \theta_2$ e θ_3. Nesta simulação também foi usado o Método da Ponderação dos Objetivos e o Método do Critério Global para resolver o problema multiobjetivo dado pela Eq. (5.46). Novamente, os resultados obtidos com a ED e EDMP foram muito semelhantes. No entanto, enquanto ED precisou em torno de 11 a 30 horas, dependendo do método, para encontrar a solução, EDMP gastou em média 14 minutos.

As bem sucedidas aplicações numéricas demonstram a excelente eficiência do método EDMP. Os resultados indicam que EDMP é mais rápido para alcançar a solução ótima e que o algoritmo é altamente paralelizável. Além disso, visto que a comunicação entre os processadores é mínima, pode-se garantir grande diminuição no tempo de execução do programa em relação ao algoritmo sequencial.

5.3.2 Problema Restrito: Otimização de Sistemas Robóticos Considerando sua Topologia

De acordo com OLIVEIRA (2011) para estudar as topologias dos manipuladores 3R é necessário conhecer as equações que as separam por meio das superfícies de singularidades. Serão considerados os manipuladores com três juntas rotacionais com eixos ortogonais ($\theta_2 = -90$ e $\theta_3 = 90$). O estudo deste tipo de manipulador se baseia na topologia das superfícies de singularidades do espaço de trabalho e é feito em função dos parâmetros de Denavit-Hartenberg restantes: d_2, d_3, d_4 e r_2. A fim de reduzir o número de parâmetros, sem perda de generalidade, podem-se normalizar todos estes parâmetros em relação à d_2 (obviamente qualquer outro parâmetro pode ser utilizado). Adotando-se $d_2 = 1$, os parâmetros geométricos considerados são d_3, d_4 e r_2. As três variáveis das juntas consideradas são: θ_1, θ_2 e θ_3.

Substituindo as hipóteses simplificadoras $\theta_2 = -90, \theta_3 = 90, r_3 = 0$ e $d_2 = 1$, na

Eq.(5.43), obtém-se as seguintes expressões:

$$\begin{aligned} x &= \left[1 + \left(d_3 + d_4 c_3\right) c_2\right] c_1 - \left(r_2 + d_4 s_3\right) s_1 \\ y &= \left[1 + \left(d_3 + d_4 c_3\right) c_2\right] s_1 + \left(r_2 + d_4 s_3\right) c_1 \\ z &= -\left(d_3 + d_4 c_3\right) s_2 \end{aligned} \tag{5.58}$$

A existência de pelo menos um ponto de singularidade no espaço de trabalho com exatamente três soluções no Modelo Geométrico Inverso (MGI) coincidentes é difícil de ser mostrada diretamente do Modelo Geométrico Direto (MGD). A ideia é eliminar as variáveis de junta, θ_1 e θ_2, do sistema, a fim de obter uma condição sobre a última variável, θ_3. Para isto, são adicionadas as relações algébricas $c\theta_i^2 + s\theta_i^2 = 1, i = 1, 2, 3$ à Eq. (5.58) do MGD, resultando num sistema algébrico de 6 equações:

$$\begin{aligned} f_1 &= x - \left[1 + \left(d_3 + d_4 c_3\right) c_2\right] c_1 + \left(r_2 + d_4 s_3\right) s_1 \\ f_2 &= y - \left[1 + \left(d_3 + d_4 c_3\right) c_2\right] s_1 - \left(r_2 + d_4 s_3\right) c_1 \\ f_3 &= z + \left(d_3 + d_4 c_3\right) s_2 \\ f_4 &= c_1^2 + s_1^2 - 1 \\ f_5 &= c_2^2 + s_2^2 - 1 \\ f_6 &= c_3^2 + s_3^2 - 1 \end{aligned} \tag{5.59}$$

Agora, para eliminar variáveis θ_1 e θ_2 calcula-se a base de Groebner para o ideal $I = f_1, f_2, \cdots, f_6$. Para isso, usa-se um programa de álgebra computacional (Maple, Axiom ou Maxima). Assim, uma base de Groebner para I é composta pelos polinômios:

$$p_1(\theta_3) = c_3^2 + s_3^2 - 1 \tag{5.60}$$

e

$$p_2(\theta_3) = m_5 c_3^2 + m_4 s_3^2 + m_3 c_3 s_3 + m_2 c_3 + m_1 s_3 + m_0 \tag{5.61}$$

onde

$$\begin{cases} m_0 = -x^2 - y^2 + r_2^2 + \frac{(R+1-L)^2}{4} \\ m_1 = 2r_2 d_4 + (L - R - 1) d_4 r_2 \\ m_2 = (L - R - 1) d_3 d_4 \\ m_3 = 2r_2 d_3 d_4^2 \\ m_4 = d_4^2 (r_2^2 + 1) \\ m_5 = d_3^2 d_4^2 \end{cases} \tag{5.62}$$

com

$$R = x^2 + y^2 + z^2 \qquad \text{e} \qquad L = d_4^2 + d_3^2 + r_2^2 \tag{5.63}$$

Considerando a nova variável $t = tg(\theta_3/2)$ segue que:

$$c\theta_3 = \frac{1 - t^2}{1 + t^2} \qquad \text{e} \qquad s\theta_3 = \frac{2t}{1 + t^2} \tag{5.64}$$

Substituindo na Eq. (5.61) e realizando algumas manipulações algébricas, obtém-se o seguinte polinômio:

$$P(t) = at^4 + bt^3 + ct^2 + dt + e \tag{5.65}$$

onde

$$\begin{aligned} a &= m_5 - m_2 + m_0 \\ b &= -2m_3 + 2m_1 \\ c &= -2m_5 + 4m_4 + 2m_0 \\ d &= 2m_3 + 2m_1 \\ e &= m_5 + m_2 + m_0 \end{aligned} \tag{5.66}$$

Um manipulador é cuspidal se o polinômio $P(t)$, que possui grau 4 e coeficientes que dependem de x, y, z, d_4, d_3 e r_2, admitir raiz real tripla. Isto equivale a resolver o seguinte sistema:

$$S(t, a, b, c, d, e) = \begin{cases} P(t, Z, R, d_3, d_4, r_2) = 0 \\ \frac{\partial P}{\partial t}(t, Z, R, d_3, d_4, r_2) = 0 \\ \frac{\partial^2 P}{\partial t^2}(t, Z, R, d_3, d_4, r_2) = 0 \end{cases} \tag{5.67}$$

onde $Z = z^2$. No sistema polinomial dado pela Eq. (5.67), as variáveis do problema são t, R e Z, enquanto os parâmetros estritamente positivos do problema são d_3, d_4 e r_2 (observe que o anulamento de um destes parâmetros resulta em um manipulador não cuspidal). Para escrever a condição dependendo somente dos parâmetros DH é necessário eliminar as variáveis t, R e Z do sistema da pela Eq. (5.67). Resolver o sistema é equivalente a dividir o primeiro quadrante do espaço dos parâmetros (o espaço $d_3 d_4 r_2$) em várias regiões em que o número das soluções do problema é constante. Para cada região, são obtidos o número de soluções do problema e um conjunto de parâmetros do manipulador que representam esta região. De acordo com as hipóteses $Z > 0$ e $R - Z > 0$, é possível eliminar as variáveis t, Z e R. Depois de várias mudanças de variáveis e utilizando a base de Grobner, segundo OLIVEIRA et al (2008), são obtidas as cinco equações abaixo:

$$\begin{aligned}
&g_1 : r_2^2 + d_3^2 - d_4^2 = 0, \\
&g_2 : d_4^2 \left[\left(1 + r_2^2 + d_3^2\right)^2 - 4 d_3^2 \right] \left(r_2^2 + d_3^2 - d_4^2\right) - r_2^2 d_3^2 = 0, \\
&g_3 : r_2^2 + d_4 - d_3 + d_4 = 0, \\
&g_4 : \left(d_3 - 1\right)^2 \left(d_3^2 - d_4^2\right) + r_2^2 d_3^2 = 0, \\
&g_5 : \left(d_3 + 1\right)^2 \left(d_3^2 - d_4^2\right) + r_2^2 d_3^2 = 0.
\end{aligned} \tag{5.68}$$

Agora tem-se condições de obter a superfície que distingue dois tipos de manipuladores: binário e quaternário. O manipulador é binário (resp. quaternário) se o MGI tem no máximo 2 (resp. 4) soluções. Um manipulador binário é sempre não cuspidal. Uma vez que a equação para g_2 tem grau 2 em d_4 é possível obter a equação de separação:

$$C_1 : d_4 = \sqrt{\frac{1}{2}\left(d_3^2 + r_2^2 - \frac{(d_3^2 + r_2^2)^2 - d_3^2 + r_2^2}{AB}\right)}, \tag{5.69}$$

onde $A = \sqrt{(d_3 + 1)^2 + r_2^2}$ e $B = \sqrt{(d_3 - 1)^2 + r_2^2}$.

As outras superfícies são obtidas anulando o determinante da matriz jacobiana J do Modelo Geométrico Direto:

$$det(J) = d_4(d_3 + d_4 c_3)[s_3 + (d_3 s_3 - r_2 c_3)c_2] \tag{5.70}$$

Assim, as superfícies de separação de C_2, C_3 e C_4 são obtidas resolvendo a Eq. (5.70) e são dadas por:

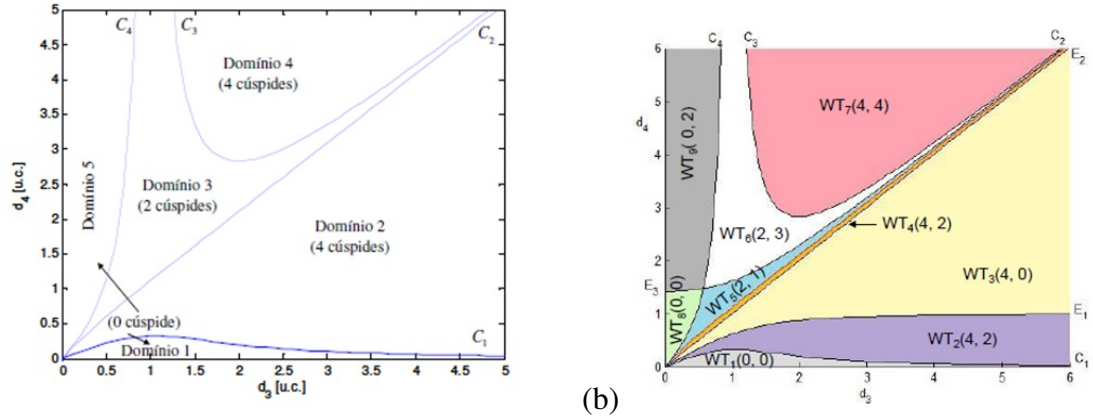
$$C_2 : d_4 = d_3/(1+d_3)\sqrt{(d_3+1)^2 + r_2^2} \tag{5.71}$$

$$C_3 : d_4 = d_3/(d_3-1)\sqrt{(d_3-1)^2 + r_2^2}, \quad \text{com } d_3 > 1 \tag{5.72}$$

$$C_4 : d_4 = d_3/(1-d_3)\sqrt{(d_3-1)^2 + r_2^2} \quad \text{com } d_3 < 1 \tag{5.73}$$

Resumindo, o espaço dos parâmetros (d_3, d_4 e r_2) de um manipulador ortogonal 3R está dividido em cinco domínios separados pelas superfícies C_1, C_2, C_3 e C_4, definidas, respectivamente, pelas Eqs. (5.69), (5.71), (5.72) e (5.73).

A Fig. 5.7a mostra as curvas de separação de uma secção plana (d_3, d_4) do espaço dos parâmetros, resultando em cinco domínios, adotando um valor fixo para r_2 = 1. A Fig. 5.7b mostra o espaço de parâmetros divididos de acordo com o número de pontos de cúspides e de pontos de nós. Os domínios, de acordo com o número de pontos de cúspide, são divididos em subdomínios que contêm o mesmo número de nós. Cada sub-domínio define a topologia do espaço de trabalho indicada por $WT_i(\alpha, \beta)$, onde α representa o número de pontos de cúspide e β o número de pontos de nós.



(a) (b)

Figura 5.7 – Divisão do espaço de parâmetros considerando $r_2 = 1$: (a) De acordo com a superfície de separação de topologias, (b) De acordo com o número de pontos de cúspides e nós. Fonte: BAILI (2004) pg. 91 e pg. 102.

A Eq. (5.69) representa a superfície de separação entre os manipuladores binário e quaternário. Os manipuladores que pertencem ao domínio 1 são binários, tem

uma cavidade toroidal na sua área de trabalho e não têm pontos de cúspide e de nó. Este manipulador, representado na Fig. 5.8a, caracteriza o primeiro tipo de manipulador, cuja topologia é conhecida como $WT_1(0,0)$.

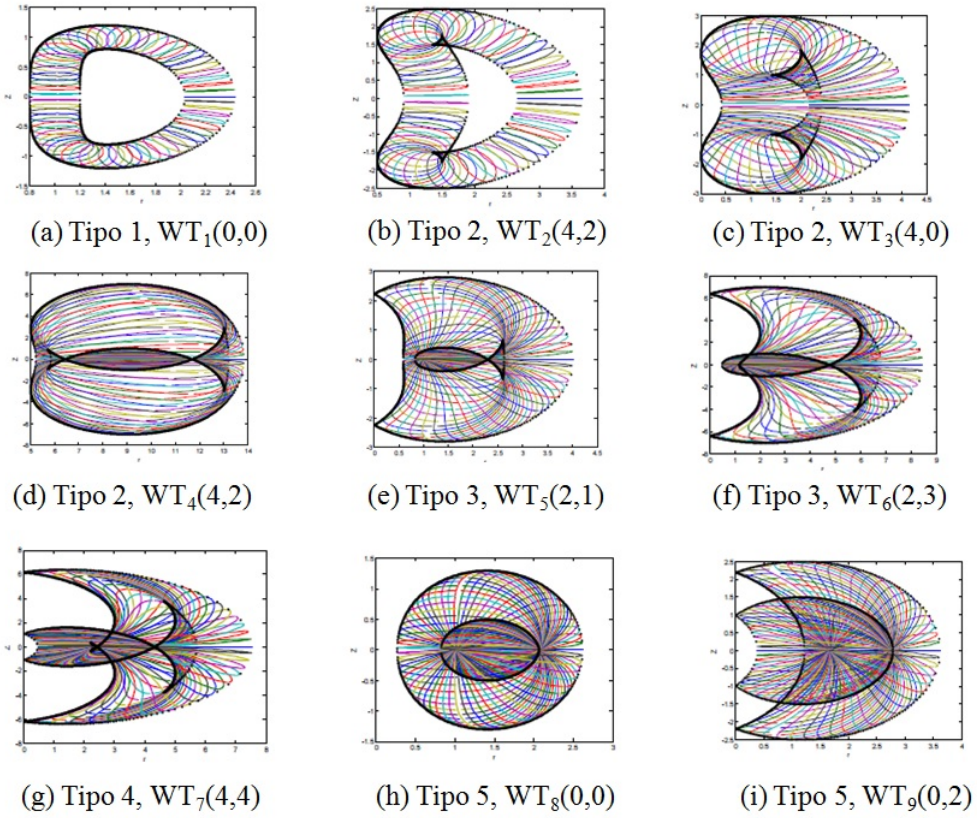


Figura 5.8 – Seção radial de manipuladores ortogonais 3R, mostrando os 5 tipos de manipuladores. Fonte: OLIVEIRA (2011).

O domínio 2 representa os manipuladores quaternários que têm 4 pontos de cúspide. Esta região pode ser subdividida em três sub-domínios através das superfícies E_1 e E_2. A topologia do espaço de trabalho $WT_2(4,2)$, representada na Fig. 5.8b, tem quatro pontos de cúspide, 2 nós, uma cavidade toroidal, duas regiões com 4 soluções e uma região com 2 soluções em MGI. A topologia $WT_3(4,0)$ contém manipuladores com 4 pontos de cúspides, zero nós, nenhuma cavidade toroidal, uma região com 4 soluções e outra com 2 soluções em MGI, como ilustrado na Fig. 5.8c. A transição entre as topologias WT_2 e WT_3 é o limite entre os manipuladores que contêm uma cavidade toroidal em seu espaço de trabalho e aqueles que não o contêm. Segundo BAILI (2004),

a superfície de separação entre as topologias é dada pela expressão:

$$E_1 : d_4 = 0,5(A - B), \tag{5.74}$$

onde A e B são dados na Eq. (5.69).

No domínio 2, ainda é possível caracterizar a topologia representada na Fig. 5.8d, denotada por $WT_4(4,2)$, que contém 4 pontos de cúspide e 2 nós. Estes nós são diferentes dos nós de WT_2 uma vez que não delimitam uma cavidade toroidal mas uma região de quatro soluções no MGI. Neste caso, a superfície de separação E_2, entre as topologias WT_3 e WT_4 é definida por:

$$E_2 : d_4 = d_3. \tag{5.75}$$

A superfície C_2 separa os domínios 2 e 3 e é dada pela Eq. (5.71). O domínio 3 é composto por manipuladores que apresentam dois pontos de cúspides no invólucro interno e pode ser dividido em dois subdomínios através da superfície E_3. Os manipuladores descritos por $WT_5(2,1)$ tem dois pontos de cúspides no invólucro interno, um ponto de nó e a forma de um peixe, como mostrado na secção radial apresentada na Fig. 5.8e. Além disso, a Fig. 5.8f apresenta a secção radial de um manipulador que pertence a topologia $WT_6(2,3)$, que tem 2 pontos de cúspide e 3 nós. A Eq. (5.78) define a superfície de separação entre as topologias WT_5 e WT_6. Além disso, esta superfície também separa a topologia dos espaços de trabalho WT_8 e WT_9 que estão contidas no domínio 5.

$$E_3 : d_4 = 0,5(A + B). \tag{5.76}$$

A superfície C_3, Eq. (5.72), separa os manipuladores dos domínios 3 e 4. No domínio 4 , os manipuladores são do tipo 4, representado por $WT_7(4,4)$, possuem 4 pontos de cúspide e 4 nós, como pode ser visto na Fig. 5.8g. Os 4 pontos de cúspide são compartilhados entre as superfícies internas e externas de singularidade.

Finalmente, o domínio 5 corresponde aos manipuladores que não possuem pontos de cúspide (Figs. 5.8h e 5.8i). No caso do domínio 5, ao contrário dos mani-

puladores do tipo 1, o invólucro interno não é definido por uma cavidade toroidal, mas por uma região com 4 soluções em MGI. A superfície C_4, representada pela Eq. (5.73), separa os domínios 3 e 5.

O domínio 5 corresponde aos manipuladores do tipo 5. Esta região é dividida em 2 subdomínios por meio da superfície E_3. Na Fig. 5.8h, a topologia representada por $WT_8(0,0)$ não tem pontos de cúspides ou nós. Como mencionado anteriormente, o invólucro interno não é definido por uma cavidade toroidal, mas por uma região com 4 soluções no MGI. Finalmente, a Fig. 5.8i apresenta um manipulador que pertence a topologia $WT_9(0,2)$, com 0 pontos de cúspide e 2 pontos de nós obtidos pela intersecção do invólucro interno e externo.

O problema de otimização é formulado com o objetivo de obter os parâmetros geométricos ideais do manipulador 3R para maximizar o espaço de trabalho e da rigidez do sistema e para minimizar a destreza para que as topologias especificadas pelo projetista sejam obedecidas. Visto que o problema se trata de um problema de otimização multiobjetivo é necessário localizar todas as compensações possíveis entre várias funções objetivos que geralmente são conflitantes entre si. As restrições dependem da topologia escolhida para o robô, de acordo com a Fig. 5.8. A otimização será investigada utilizando Algoritmos Genéticos (GA), Evolução Diferencial (DE) e Evolução Diferencial Melhorada implementada em paralelo (EDMP).

Os algoritmos evolutivos foram desenvolvidos para problemas sem restrições. Assim, no caso de problemas de otimização com restrições, é necessário introduzir modificações no método. Será utilizado o conceito de Função de Penalidade de NOCEDAL e WRIGHT (2000). Foi adotado $r_p = 1000$. Então, o problema pode ser reescrito como se segue:

$$\text{Max } F(x) = f(x) + r_p P(x), \text{ onde } f(x) = [V, D, R] \text{ e } P(x) = \max(0, g_j(x))^2. \quad (5.77)$$

Sujeito à : $g_j(x) \leq 0;\ j = 1, \cdots, k$ e $x^l \leq x_i \leq x^u,\ i = 1, 2, 3$.

Os parâmetros geométricos são as variáveis de projeto dadas por $x = (d_3, d_4, r_2)^T$. Os limites inferiores e superiores adotados para o comprimento do braço (restrições la-

terais) são: $0,1 \leq x_i \leq 3,0,\ i = 1,2,3$.

Nesta simulação, dois métodos de otimização multiobjetivo são utilizados: Método dos Objetivos Ponderados e Método do Critério Global (métrica L_{2r} e métrica L_{3r}) apresentado por OLIVEIRA e SARAMAGO (2010).

A estratégia de soma ponderada converte o vetor $f(x)$ de um problema multiobjetivo em um problema de otimização escalar através de uma soma ponderada de todos os objetivos como Eq. (5.78). Os coeficientes de ponderação w_i representam a importância relativa de cada critério. Assim,

$$\text{Maximizar } F(x) = w_1 Vc_1 - w_2 Dc_2 + w_3 Rc_3 - r_p P(x), \text{ onde } \sum_{i=1}^{3} w_i = 1, \qquad (5.78)$$

sendo que o volume do espaço de trabalho V é dada pela Eq. (5.48), a rigidez R é calculada utilizando a Eq. (5.56) e a destreza D é representada na Eq.(5.57).

A ponderação dos objetivos é o modelo substituto mais comum para problemas de otimização vetoriais. A dificuldade aqui é anexar os coeficientes de ponderação para cada um dos objetivos. Os coeficientes de ponderação não necessariamente correspondem diretamente à importância relativa dos objetivos ou permitem que os *tradeoffs* entre os objetivos sejam expressos. Para que w_i na Eq. (5.78) possa refletir estreitamente a importância de objetivos todas as funções devem ser expressas em unidades que aproximadamente representam os mesmos valores numéricos. Os melhores resultados são geralmente obtidos quando $c_i = 1/f_i^0$, onde f_i^0 representa a solução ideal, que indica o valor mínimo de cada i-ésima função. Para determinar esta solução, deve-se encontrar o mínimo possível para todas as funções objetivo separadamente. Neste caso, o vetor $f^0 = [V_{id}, D_{id}, R_{id}]^T$ é o vetor dos valores ideais de um problema de otimização multiobjetivo.

No Método do Critério Global, o problema de otimização multiobjetivo é transformado em um problema de otimização escalar usando um critério global. A função que descreve este critério global pode ser definida como uma possível solução próxima da solução ideal encontrada. Neste caso, a métrica-L_{2r} e a métrica-L_{3r}, são dadas respecti-

vamente por:

$$\text{Minimizar } F(x) = \left(\left(\frac{V_{id} - V}{V_{id}} \right)^2 + \left(\frac{D_{id} - D}{D_{id}} \right)^2 + \left(\frac{R_{id} - R}{R_{id}} \right)^2 \right)^{1/2} + r_p P(x). \quad (5.79)$$

$$\text{Minimizar } F(x) = \left(\left| \frac{V_{id} - V}{V_{id}} \right|^3 + \left| \frac{D_{id} - D}{D_{id}} \right|^3 + \left| \frac{R_{id} - R}{R_{id}} \right|^3 \right)^{1/3} + r_p P(x). \quad (5.80)$$

Para este problema foram adotados os seguintes parâmetros:

- ED: número de indivíduos na população N_p = 15; 100 gerações, fator de diferença ponderada F = 0,8 e probabilidade de cruzamento CR = 0,5.
- AG: N_p = 80 indivíduos, 100 gerações, probabilidades de cruzamento e mutação: 0,60 e 0,02.
- EDMP: população com 64 indivíduos divididos em 4 processadores, 100 gerações, fator de diferença F = 0,8 e probabilidade de cruzamento CR = 0,6. O critério de parada do algoritmo EDMP adotado foi o número máximo de gerações da população e a verificação de sua estagnação. O processo de optimização é interrompido quando não ocorre uma melhoria significativa no valor da função objetivo após 30 iterações. Isto é, se a seguinte comparação $|f_k - f_{k+1}| < 10^{-6}$ ocorre 30 vezes e as 100 iterações ainda não foram completadas, o algoritmo para. As simulações foram desenvolvidas em um computador Intel® Core^{TM} i5-430M e 6 GB de RAM.

Exemplo 1 - Manipuladores da Topologia $WT_1(0,0)$

Neste exemplo, será considerado uma aplicação onde o manipulador deve pertencer a topologia WT_1 (ver Fig. 5.8b). Neste caso, as seguintes restrições laterais devem ser obedecidas: 0,1 < d_3, d_4, r_2 < 3,0 [u.c.].

Os pontos pertencem à curva C_1, dada pela Eq. (5.69). As soluções ideais calculadas usando ED foram: V_{id} = 99,7175 [u.v.]; D = 1,1979 e R_{id} = 35,8325 [u.r.].

Os resultados ótimos obtidos através do Método da Ponderação dos Objetivos, Eq. (5.78), são apresentados na Tab. 5.14. Observando esta tabela, pode-se

notar que a melhor solução depende do interesse do projetista, pois cada função objetivo está conflitando com a outra. Neste exemplo, quando se adotou os coeficientes de ponderação igual a 0,8 para o volume (w_1) ou para a rigidez (w_3), obteve-se resultados semelhantes. Isto é devido ao fato de que ambos são maximizados e apresentaram um comportamento semelhante. Mas ao adotar o coeficiente de ponderação igual a 0,8 para a destreza (w_2) observou-se que um resultado diferente foi obtido e a destreza foi significativamente melhorada. Os resultados indicam que este problema é sensível ao valor da destreza. Quando esta função é priorizada, os valores ótimos do volume e da rigidez são fortemente modificados.

Tabela 5.14 – Valores ótimos considerando o método da Ponderação dos Objetivos para o exemplo 1.

Coef. de Poderação w_i	Algoritmo	$[d_3\ \ d_4\ \ r_2]$ [u.c.]	Volume [u.v.]	Destreza	Rigidez [u.r.]	Tempo (min)	N_{ava}
$w_1 = 0,33$ $w_2 = 0,33$ $w_3 = 0,33$	AG$^{\Delta}$	0,65 0,44 0,52	25,46	1,19	6,05	117,7	
	ED$^{\Delta}$	0,97 0,73 0,10	58,63	1,87	10,72	67,34	
	EDMP	0,92 0,45 0,61	39,18	1,41	8,15		
		0,94* 0,45* 0,61*	39,67*	1,42*	8,23*	0,55	656
		0,02** 0,001** 0,002**	0,85**	0,03**	0,14**		
$w_1 = 0,10$ $w_2 = 0,80$ $w_3 = 0,10$	AG$^{\Delta}$	0,60 0,43 0,49	22,64	1,18	5,63	89,12	
	ED$^{\Delta}$	0,60 0,43 0,48	22,68	1,18	5,63	60,88	
	EDMP	0,70 0,44 0,56	28,17	1,21	6,49		
		0,67* 0,44* 0,54*	26,70*	1,21*	6,26*	0,54	736
		0,02** 0,001** 0,01**	1,00**	0,004**	0,15**		
$w_1 = 0,10$ $w_2 = 0,10$ $w_3 = 0,80$	AG$^{\Delta}$	1,00 0,44 0,61	42,57	1,51	8,65	121,9	
	ED$^{\Delta}$	0,97 0,74 0,10	59,14	1,89	10,79	56,39	
	EDMP	0,93 0,79 0,10	59,93	1,99	11,01		
		1,01* 0,61* 0,34*	52,37*	1,86*	10,09*	0,52	704
		0,07** 0,18** 0,24**	7,57**	0,13**	0,91**		

$^{\Delta}$ Fonte: OLIVEIRA (2011), * Média dos valores, ** desvio padrão e N_{ava} número de avaliações da função objetivo.

Tabela 5.15 – Valores ótimos considerando o método do Critério Global para o exemplo 1.

Métrica	Algoritmo	$[d_3\ \ d_4\ \ r_2]$ [u.c.]	Volume [u.v.]	Destreza	Rigidez [u.r.]	Tempo (min)	N_{ava}
L_{2r}	AG$^{\Delta}$	0,97 0,45 0,59	41,73	1,45	8,46	87,39	
	ED$^{\Delta}$	0,95 0,45 0,59	40,87	1,42	8,35	65,17	
	EDMP	0,84 0,45 0,61	35,05	1,30	7,51		
		0,83* 0,45* 0,60*	34,50*	1,29*	7,43*	0,57	736
		0,01* 0,001* 0,002*	0,54**	0,001**	0,07**		
L_{2r}	AG$^{\Delta}$	0,92 0,54 0,41	44,33	1,67	8,66	75,45	
	ED$^{\Delta}$	0,97 0,73 0,10	58,64	1,87	10,72	41,52	
	EDMP	0,76 0,45 0,58	31,15	1,24	6,93		
		0,75* 0,45* 0,58*	30,54*	1,23*	6,84*	0,52	672
		0,01* 0,002* 0,006*	0,49**	0,005**	0,07**		

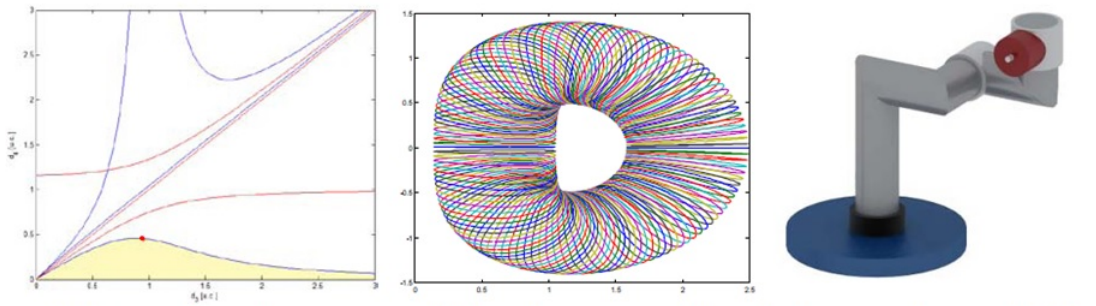
$^{\Delta}$ Fonte: OLIVEIRA (2011), * Média dos valores, ** desvio padrão e N_{ava} número de avaliações da função objetivo.

Os valores calculados usando o Método do Critério Global, dada nas Eqs. (5.79) e (5.80), podem ser observados na Tab. 5.15. Nesta técnica, a ideia consiste em minimizar o erro relativo das funções em relação aos valores ideais. As soluções obtidas representam um compromisso entre as três funções objetivos.

Os valores encontrados com os algoritmos AG e ED apresentados nas Tabs. 5.14 e 5.15 são de OLIVEIRA (2011).

Observando as Tabs. 5.14 e 5.15 pode-se observar que ambas as técnicas encontraram valores semelhantes, diferindo apenas no custo computacional.

Considerando a métrica-L_{2r}, o ponto ótimo obtido pelo método Evolução Diferencial está marcado na Fig. 5.9a. A área da secção radial ótima do espaço de trabalho é apresentado na Fig. 5.9b. Comparando a secção radial da Fig. 5.9b com a Fig. 5.8a, pode-se notar que os parâmetros projetados resultam em um manipulador com um volume maior (o vazio da área de trabalho foi reduzida). O manipulador ótimo pertencente a topologia WT_1(d_2 = 1, r_3 = 0) é apresentado na Fig. 5.9c.



(a) Espaço de parâmetros com d_2=1, r_3=0.

(b) Área da secção radial do espaço de trabalho.

(c) Esquema de um robô 3R.

Figura 5.9 – Projeto ótimo de um robô 3R considerando a métrica-L_{2r} pela Evolução Diferencial - Exemplo 1.

Exemplo 2 - Manipuladores da Topologia $WT_3(4,0)$

Agora, considerando um manipulador que pertence a topologia WT_3, são adotadas as seguintes restrições: 0,1 < d_3, d_4, r_2 < 3,0 [u.c.].

Pontos acima da curva E_1, dada pela Eq. (5.74) e pontos abaixo da curva E_2, dada pela Eq. (5.75). As soluções ideais calculadas utilizando ED foram: V_{id}= 1896,784 [uv]; D_{id} = 1,018 e R_{id} = 105,66 [us]. Para este caso, os melhores resultados obtidos pelo Método da Ponderação dos Objetivos são mostrados na Tab. 5.16 e a Tab. 5.17 apresenta as soluções ótimas calculadas usando o Método do Critério Global. Como observado no Exemplo 1, a melhor solução depende de interesse do projetista.

Considerando-se a Métrica L_{2r}, o ponto ótimo obtido por Evolução Diferencial

Tabela 5.16 – Valores ótimos considerando o método da Ponderação dos Objetivos para o exemplo 2.

Coef. de Poderação w_i	Algoritmo	$[d_3\ \ d_4\ \ r_2]$ [u.c.]	Volume [u.v.]	Destreza	Rigidez [u.r.]	Tempo (min)	N_{ava}
$w_1 = 0,33$ $w_2 = 0,33$ $w_3 = 0,33$	AG$^{\Delta}$	3,00 3,00 3,00	1896,78	1,018	105,66	55,03	
	ED$^{\Delta}$	3,00 3,00 3,00	1896,78	1,018	105,66	12,87	
	EDMP	3,00 3,00 3,00	1887,62	1,02	105,66		
		3,00* 3,00* 3,00*	1887,62*	1,02*	105,66*	0,43	560
		0** 0** 0**	1,3e-12**	1,3e-15**	2,8e-14**		
$w_1 = 0,80$ $w_2 = 0,10$ $w_3 = 0,10$	AG$^{\Delta}$	3,00 3,00 3,00	1896,78	1,018	105,66	114,02	
	ED$^{\Delta}$	3,00 3,00 3,00	1896,78	1,018	105,66	66,37	
	EDMP	3,00 3,00 3,00	1887,62	1,018	105,66		
		3,00* 3,00* 3,00*	1887,62*	1,02*	105,66*	0,41	528
		0** 0** 0**	1,3e-12**	1,3e-15**	2,8e-14**		
$w_1 = 0,10$ $w_2 = 0,80$ $w_3 = 0,10$	AG$^{\Delta}$	3,00 2,97 2,94	1850,93	1,010	104,12	114,56	
	ED$^{\Delta}$	3,00 3,00 3,00	1892,50	1,017	105,59	9,80	
	EDMP	3,00 3,00 3,00	1887,63	1,017	105,66		
		3,00* 3,00* 3,00*	1887,62*	1,02*	105,66*	0,40	512
		0** 0** 0**	1,3e-12**	1,3e-15**	2,8e-14**		
$w_1 = 0,10$ $w_2 = 0,10$ $w_3 = 0,80$	AG$^{\Delta}$	3,00 3,00 3,00	1896,78	1,018	105,66	119,84	
	ED$^{\Delta}$	3,00 3,00 3,00	1896,78	1,018	105,66	55,21	
	EDMP	3,00 3,00 3,00	1887,62	1,018	105,66		
		3,00* 3,00* 3,00*	1887,62*	1,02*	105,66*	0,40	528
		0** 0** 0**	1,3e-12**	1,3e-15**	2,8e-14**		

$^{\Delta}$ Fonte: OLIVEIRA (2011), * Média dos valores, ** desvio padrão e N_{ava} número de avaliações da função objetivo.

é apresentado na Fig. 5.10a. O esquema do manipulador ótimo pertencente a topologia WT_3 está ilustrado na Fig. 5.10b.

A área da secção radial ótima do espaço de trabalho é apresentado na Fig. 5.10b. Comparando a secção radial da Fig. 5.10b com a Fig. 5.8c pode-se observar que o espaço de trabalho é maior. Além disso, os manipuladores deste tipo de topologia permanecem com 4 pontos de cúspides, uma região com 4 soluções e uma outra com 2 soluções em Modelo Geométrico Inverso.

No exemplo 1, observe que os baixos valores encontrados em relação aos valores ideais, são devido às dificuldades impostas pela restrição C_1. No exemplo 2, os valores não sofrem muitas alterações devido às restrições serem mais simples.

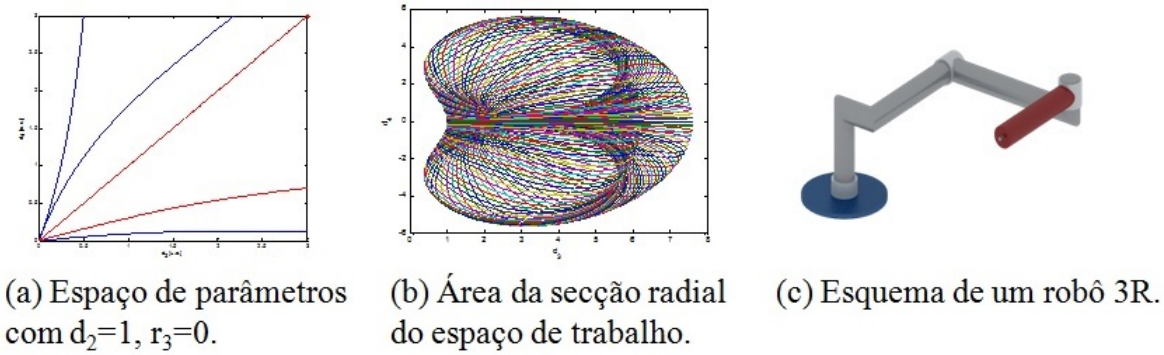
É importante observar que as metodologias aplicadas foram eficazes para obter as melhores soluções obedecendo as restrições de topologia.

Novamente, o algoritmo EDMP mostrou-se muito eficiente. Os resultados mostram que EDMP é mais rápido na busca pela solução ótima. Entre os métodos de otimização discutidos aqui, a saber, Algoritmo Genético, Evolução Diferencial e Evolução Diferencial Melhorada implementada em paralelo, a metodologia EDMP apresentou os melhores resultados levando em conta o esforço computacional, uma vez que foram necessários apenas alguns segundos para obter a solução ótima, enquanto os outros

Tabela 5.17 – Valores ótimos considerando o método do Critério Global para o exemplo 2.

Métrica	Algoritmo	$[d_3\ \ d_4\ \ r_2]$ [u.c.]	Volume [u.v.]	Destreza	Rigidez [u.r.]	Tempo (min)	N_{ava}
L_{2r}	AG$^{\Delta}$	3.00 3.00 3.00	1896.78	1.018	105.66	59.78	
	ED$^{\Delta}$	3.00 3.00 3.00	1896.78	1.018	105.66	13.35	
	EDMP	3.00 3.00 2.98	1877.21	1.016	105.48		
		3.00* 3.00* 2.99*	1880.76*	1.02*	105.51*	0.40	512
		0** 0.002** 0.01**	7.25**	0.001**	0.17**		
L_{3r}	AG$^{\Delta}$	3.00 3.00 3.00	1896.77	1.018	105.66	66.28	
	ED$^{\Delta}$	3.00 3.00 3.00	1896.78	1.018	105.66	11.93	
	EDMP	3.00 2.96 2.93	1833.32	1.009	103.82		
		3.00* 2.95* 2.93*	1831.42*	1.01*	103.74*	0.43	528
		0.002** 0.003** 0.002**	4.78**	9.7e-05**	0.17**		

$^{\Delta}$ Fonte: OLIVEIRA (2011), * Média dos valores, ** desvio padrão e N_{ava} número de avaliações da função objetivo.



(a) Espaço de parâmetros com d_2=1, r_3=0. (b) Área da secção radial do espaço de trabalho. (c) Esquema de um robô 3R.

Figura 5.10 – O projeto ideal de um robô 3R, considerando Métrica L_{2r} usando Evolução Diferencial - Exemplo 2.

métodos precisaram de minutos ou de horas para encontrá-la.

5.4 Resolução de Grandes Sistemas Lineares

A descrição e a solução de sistemas lineares são de grande importância em diversas áreas das ciências aplicadas. Segundo LEON (2011), mais de 75% dos problemas matemáticos encontrados em aplicações científicas e industriais envolvem a resolução de sistemas lineares em algum estágio. No entanto, estes sistemas geralmente possuem dimensão elevada o que dificulta a obtenção da sua solução de forma algébrica. De forma geral, os sistemas lineares podem ser resolvidos pelos métodos diretos e indiretos.

Os métodos diretos são aqueles que após um número finito e bem determinado de operações obtém-se a solução exata, a menos de erros de arredondamento, de um sistema linear. Para resolver um sistema linear por meio de um método direto é necessário modificar a matriz original dos coeficientes. Por isso, caso esta matriz seja esparsa

estes métodos podem destruir sua esparsidade, no todo ou em parte.

Os métodos indiretos ou iterativos são aqueles que, a partir de uma aproximação inicial, obtém-se a solução aproximada do sistema linear por meio de um processo iterativo. Esta solução aproximada depende de uma tolerância prefixada. Os métodos iterativos são classificados em estacionários e não-estacionários. Nos métodos estacionários cada solução aproximante é obtida a partir da solução anterior aplicando-se sempre o mesmo processo, ou seja, os operadores matriciais são constantes para todas as iterações. Nos métodos não-estacionários a matriz de iteração não é constante e por isso a cada iteração procura-se obter a melhor aproximação da solução de acordo com certas restrições e utilizando informações das iterações anteriores. Estes métodos se baseiam na teoria de otimização, procurando o mínimo de um funcional em uma direção de busca bem definida. A maioria destes métodos trata apenas de sistemas lineares onde a matriz dos coeficientes é simétrica e positiva definida. Existem algumas técnicas que podem ser utilizadas nos métodos iterativos para acelerar a convergência, por exemplo, o pré-condicionamento da matriz de coeficientes, que consiste em determinar uma matriz não singular, de forma que o novo sistema possua uma taxa de convergência maior (PURCINA (2010)).

Um sistema de equações lineares é uma coleção finita de variáveis e equações lineares (todas nas mesmas variáveis), consideradas em conjunto e normalmente apresentadas na forma matricial:

$$Ax^* = b, \tag{5.81}$$

onde A $\in \mathbb{R}^{n\times n}$ é a matriz dos coeficientes, b $\in \mathbb{R}^n$ é o vetor independente e x* é o vetor das incógnitas ou vetor solução, a ser determinado. Esta equação pode ser reestruturada como:

$$r(x) = b - Ax, \tag{5.82}$$

na qual o vetor solução x* foi substituído por um vetor genérico x e por isso há um vetor resíduo $r(x)$. É claro que, quando x = x* o resíduo será nulo, $r(x) = 0$.

O objetivo é aplicar técnicas heurísticas para encontrar a solução de sistemas lineares sem se preocupar com as características da matriz de coeficientes, mesmo que

a mesma seja não simétrica e nem positiva definida. Além disso, não será necessária a utilização de pré-condicionadores para adequar o sistema aos métodos tradicionais. A metodologia utilizada consiste em minimizar o vetor resíduo Eq. (5.82) do sistema linear. Muitos métodos de otimização podem ser usados para resolver este problema de minimização. O algoritmo desenvolvido para a Evolução Diferencial Melhorada, implementada em paralelo, será aplicado na solução de grandes sistemas lineares.

5.4.1 Problema de Identificação de Forças Dinâmicas

Um processo de identificação de forças pode ser conduzido por meio de um modelo matemático que relacione as respostas medidas de um sistema mecânico com as possíveis forças aplicadas sobre o mesmo.

Os modelos de resposta são construídos a partir de equações diferencias que relacionam as entradas com as saídas do sistema em observação.

Os modelos de respostas podem ser estabelecidos por meio de modelagens nas quais os valores das saídas do sistema são obtidas por meio de experimentos com valores das entradas conhecidas. Assim, não é necessário ter conhecimento sobre a estrutura física interna do sistema.

O problema inverso da identificação de forças consiste em estimar as forças aplicadas sobre um sistema levando em consideração um modelo matemático e as respostas observadas. O modelo matemático inverso é determinado invertendo as equações matemáticas resultantes da modelagem do sistema físico.

Durante a modelagem de processos de identificação de forças é comum encontrar sistemas lineares mal condicionados que muitas vezes não podem ser resolvidos pelos métodos convencionais. Este fato justifica o uso de técnicas heurísticas para obter a solução de tais sistemas que possuem grande relevância na engenharia.

Serão considerados nesta seção dois problemas apresentados por PURCINA (2010): um teórico e um experimental. No problema teórico o sistema é excitado por uma força harmônica. Em seguida um problema real de identificação indireta de forças é resolvido considerando os dados experimentais.

As simulações com EDMP, desta seção, foram executadas no cluster do La-

boratório de Computação Científica Aplicada e Tecnologia de Informação - LCCATI da FACIP/UFU que possui as seguintes características:

- Frontend: 16 núcleos de processamento, 64GB de memória RAM, 1 adpatador Infiniband QDR X4 (40 Gbps) e duas unidades RAID (3TB home e 12TB tmp1) que são exportadas para todo o cluster;
- Oito (08) Nós computacionais, cada um com 64 núcleos de processamento, 128 GB de memória RAM, 2 adaptadores Infiniband QDR X4 (40 Gbps cada adaptador). Performance de pico de 8.8 GFlops por núcleo (4.5 TFLOPS total + 0.25 TFLOPS no frontend disponíveis para pequenas tarefas);
- Um nó de serviço com 8 núcleos, 64GB de memória RAM, 1 adpatador Infiniband QDR X4 (40 Gbps);
- Um servidor adicional com 16 núcleos de processamento, 64GB de memória RAM, 1 adpatador Infiniband QDR X4 (40 Gbps) e uma unidade de storage com 28 TB (Por enquanto este servidor está inativo);
- Switch infiniband 36 portas QDR X4 (40 Gbps por porta);
- Switch ethernet 1Gb 48 portas.

Sistema excitado por uma força harmônica

O objetivo é resolver o seguinte sistema linear:

$$X = HF \tag{5.83}$$

onde

$$X = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{Bmatrix}; \quad F = \begin{Bmatrix} f_1 \\ f_2 \\ \vdots \\ f_p \end{Bmatrix} \quad e \quad H = \begin{bmatrix} h_0 & 0 & \cdots & 0 \\ h_1 & h_0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h_{p-1} & h_{p-2} & \cdots & h_0 \end{bmatrix} \tag{5.84}$$

sendo que,

$$h(i,j) = A_0 \left(e^{-\xi\omega t(j+1)}\right) sen\left(\sqrt{1-\xi^2}\omega t(j+1)\right) \tag{5.85}$$

com i, j= 1, 2, $\cdots$, N-1, e $j \leq i$.

Considerando $A_0 = e^{-3}$, $\xi = 0,08$, $\omega = 2\pi 500$, dt= 1/8192, $t \in [0,1]$ e N = comprimento(t). A força é dada por:

$$F = 5sen(2\pi 300t) + 1,5cos(2\pi 300t) \tag{5.86}$$

Visto que se conhece a matriz H e o vetor X, o sistema linear (5.83) será resolvido usando EDMP e o método iterativo Resíduo Mínimo Generalizado (GMRES) para encontrar o vetor F. A função objetivo é dada pelo resíduo $r = |X - HF|$. A modelagem completa deste problema de identificação de forças dinâmicas pode ser visto em PURCINA (2010).

Segundo SAAD e SCHULTZ (1986), o GMRES é um método iterativo utilizado para resolver sistemas lineares que tem a propriedade de minimizar a cada iteração a norma do vetor residual sobre o subespaço de Krylov. O algoritmo é derivado a partir do processo de Arnoldi por construção da base orthogonal L_2 do subespaços de Krylov. O GMRES pode ser considerado como uma generalização do algoritmo MINRES de Paige e Saunders e é teoricamente equivalente ao método do Resíduo Conjugado Generalizado (GCR). Uma descrição mais detalhada do GMRES pode ser vista em PURCINA (2010).

A fim de avaliar o algoritmo EDMP foram realizados três testes com diferentes quantidades de indivíduos na população com os seguintes parâmetros: fator de diferença F = 0,7 e probabilidade de cruzamento CR = 0,9. O critério de parada do algoritmo EDMP adotado foi o número máximo de gerações da população e a verificação de sua estagnação. Nos três testes foram utilizados 80 processadores do cluster. Em seguida, as soluções numéricas encontradas com EDMP serão comparadas com as soluções analíticas e as obtidas pelo GMRES no MATLAB®.

Teste 1

No primeiro teste foram adotados $t \in [0; 0,01]$, 1600 indivíduos na população inicial o que equivale à 20 indivíduos por processador e, no máximo, 100 iterações para o critério de parada. Neste caso a matriz H tem 82×82 entradas. Foram executadas 114.560 avaliações da função objetivo em 6,95 minutos. A Fig. 5.11 apresenta a comparação entre as soluções obtidas e o erro relativo entre a solução analítica e a encontrada com EDMP. O valor do resíduo na solução ótima encontrada por EDMP foi $r = |X - HF| = 9,91 \times 10^{-5}$.

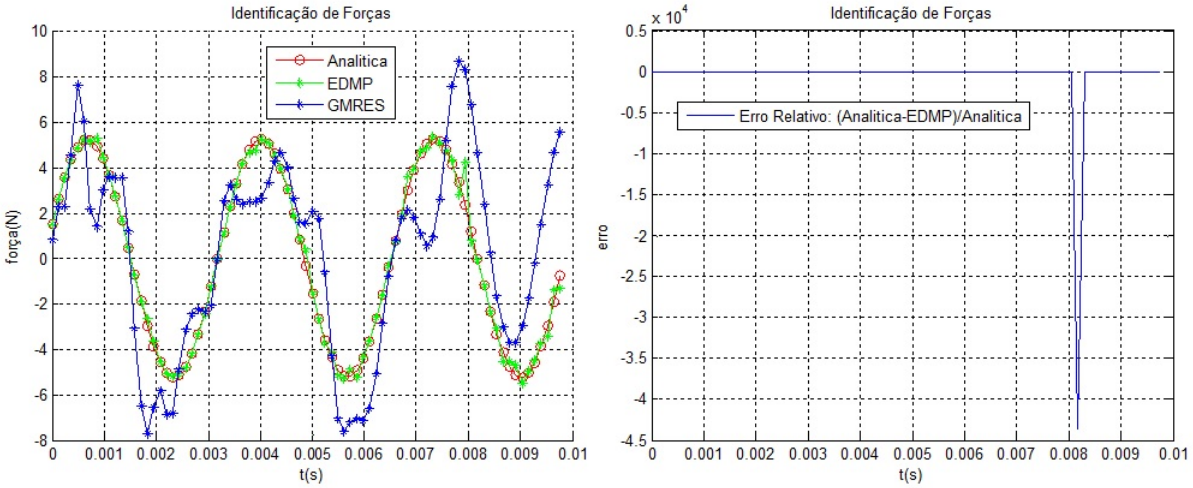


Figura 5.11 – Solução analítica e numérica encontrada com EDMP e GMRES - Teste 1.

Teste 2

A fim de tentar melhorar a solução obtida no teste 1, uma nova simulação foi executada considerando 4800 indivíduos na população inicial, o que equivale à 60 indivíduos por processador e, no máximo, 200 iterações para o critério de parada. Foram executadas 285.660 avaliações da função objetivo em 43,84 minutos. A Fig. 5.12 apresenta a comparação entre as soluções obtidas e o erro relativo entre a solução analítica e a encontrada com EDMP. O valor do resíduo na solução ótima encontrada por EDMP foi $r = |X - HF| = 9,60 \times 10^{-5}$.

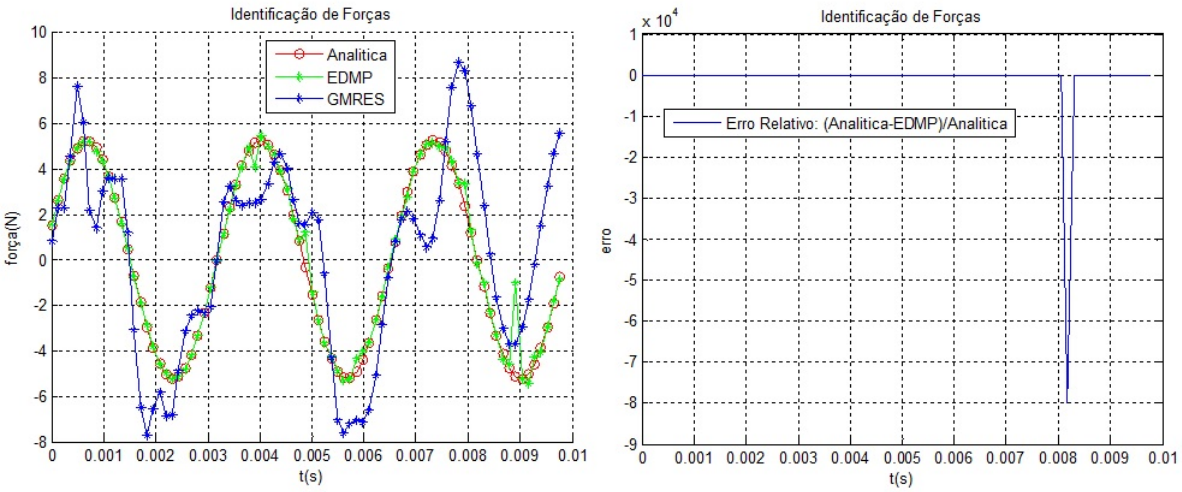


Figura 5.12 – Solução analítica e numérica encontrada com EDMP e GMRES - Teste 2.

Analizando os resultados obtidos com os testes 1 e 2 pode-se observar que as respostas dos valores da força F encontradas ao longo do tempo são muito satisfatórias já que estão muito próximas às respostas obtidas por meio da solução analítica. O erro relativo entre as soluções analítica e EDMP também mostrou-se muito pequeno, salvo em alguns pequenos intervalos onde a solução aproximada por EDMP afastou-se da solução analítica.

Há poucas diferenças entre as soluções encontradas com os testes 1 e 2. O valor do resíduo r é praticamente o mesmo, no entanto, no teste 2, apesar de haver três vezes mais indivíduos e 100 iterações a mais que o teste 1, o valor do resíduo é sutilmente maior, isto é, a diferença em módulo entre os valores dos resíduos é inferior à 0,0000032.

Portanto, pelo menos para este problema, é inviável o uso de uma população muito grande e de muitas iterações, pois tais escolhas sobrecarregam desnecessariamente os processadores e aumentam significativamente o tempo de execução do algoritmo.

Teste 3

Neste teste o objetivo é comparar as soluções entre EDMP e GMRES em simulações onde o sistema linear é “grande”. Para tanto, foi utilizado $t \in [0; 0,03]$, 3200

individuos da população inicial, o que equivale à 40 indivíduos por processador e, no máximo, 100 iterações para o critério de parada. A matriz H tem 246×246 entradas e foram executadas 557.680 de avaliações da função objetivo em 1,7 horas. A Fig. 5.13 apresenta a comparação entre as soluções obtidas e o erro relativo entre a solução analítica e a encontrada com EDMP. O valor do resíduo na solução ótima encontrada por EDMP foi $r = |X - HF| = 9,82 \times 10^{-5}$.

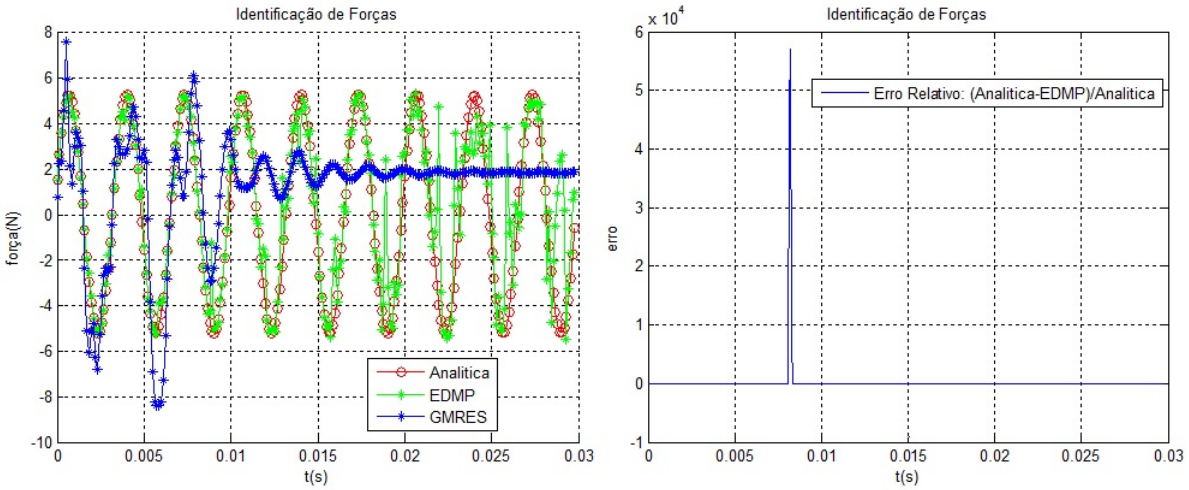


Figura 5.13 – Solução analítica e numérica encontrada com EDMP e GMRES - Teste 3.

Este teste exemplifica uma das dificuldades do GMRES que é obter a solução de “grandes” sistemas lineares. A partir de determinado momento o método começa a divergir.

A solução obtida por EDMP no teste 3 é muito mais precisa que a do GMRES e o valor do resíduo encontrado mostrou-se baixo. Uma desvantagem é o tempo de execução do algoritmo. Para esta simulação o GMRES levou 0,7 segundos para encontrar a solução do sistema porém, tal solução é bastante diferente da solução analítica. Embora o esforço computacional do EDMP seja grande, acarretando em muito tempo de execução, a solução por ele obtida está próxima da solução analítica.

Na área científica há várias pesquisas onde os sistemas lineares são representados por milhares ou até mesmo milhões de variáveis. No trabalho de DEKKER; HOFFMANN; POTMA (1994), por exemplo, é resolvido um sistema linear da ordem de 55.296 que precisou de mais de 4 dias para obter a solução por meio do método Gradi-

ente Conjugado implementado em paralelo.

Os testes realizados com EDMP permitem concluir que o método proposto se apresenta como uma alternativa para resolução de sistemas lineares de grande porte.

Na próxima subseção será resolvido um sistema linear com 3.500 variáveis para avaliar o desempenho do algoritmo EDMP na obtenção da solução ótima.

5.4.2 Sistema Dinâmico Usando Dados Experimentais

Nesta simulação o sistema linear deriva de um problema de identificação indireta de forças. O problema consiste em determinar a força aplicada a um sistema formado por três placas conectadas por três conjuntos de quatro lâminas de aço dispostas em paralelo conforme apresentado por PURCINA (2010). No domínio de baixas frequências o sistema pode ser modelado como um sistema de três graus de liberdade pouco amortecido. A Fig. 5.14 ilustra o sistema mecânico utilizado nos experimentos.

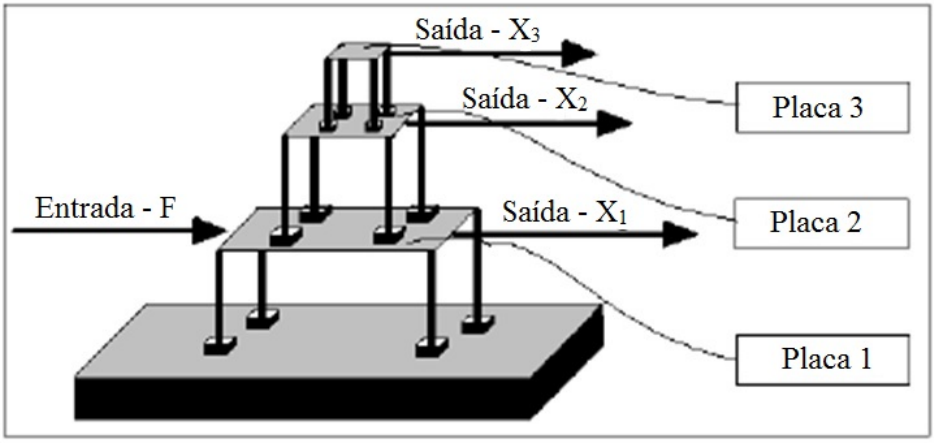


Figura 5.14 – Ilustração do sistema mecânico utilizado nos experimentos. Fonte: PURCINA (2010) pg. 119.

Neste experimento uma força de excitação por impacto F é aplicada na coordenada 1 (Placa 1). As correspondentes respostas temporais (X_1, X_2 e X_3) foram medidas nas coordenadas 1, 2 e 3. Tanto a força como as respostas temporais foram medidas somente na direção horizontal. Mais detalhes do experimento pode ser visto em PURCINA (2010).

Seja a integral de convolução de Duhamel dada pela Eq. (5.87):

$$x(t) = \int_{-\infty}^{\infty} [h(t-\tau)]\{f(\tau)\}d\tau \tag{5.87}$$

onde $h(t)$ é a F.R.I., $f(t)$ é a força excitadora em função do tempo t e $x(t)$ é a amplitude da vibração em resposta no tempo t.

Como no experimento há três locais de medição e uma força de excitação, a Eq. (5.87) pode ser desenvolvida da seguinte forma:

$$\begin{aligned} x_1(t) &= \textstyle\int_0^t [h_{11}(t-\tau)]\{f(t)\}d\tau \\ x_2(t) &= \textstyle\int_0^t [h_{21}(t-\tau)]\{f(t)\}d\tau \\ x_3(t) &= \textstyle\int_0^t [h_{31}(t-\tau)]\{f(t)\}d\tau \end{aligned} \tag{5.88}$$

sendo que a F.R.I. $h_{i1}(t)$ representa as respostas na coordenada i devido à excitação aplicada na placa 1.

A discretização Δt do tempo de amostragem da Eq. (5.88) é dada por:

$$\begin{aligned} x_1(k\Delta t) &= \textstyle\sum_{i=0}^{k} h_{11}[(k-i)]f(i)\Delta t \\ x_2(k\Delta t) &= \textstyle\sum_{i=0}^{k} h_{21}[(k-i)]f(i)\Delta t \\ x_3(k\Delta t) &= \textstyle\sum_{i=0}^{k} h_{31}[(k-i)]f(i)\Delta t \end{aligned} \tag{5.89}$$

com $k = 0, 1, 2, \cdots, p-1$.

As Eqs. (5.89) serão agrupadas em um sistema de $3p$ equações e p incógnitas, da seguinte forma matricial:

$$\{X\} = [h]\,\{F\} \tag{5.90}$$

onde

$$\{X\} = \begin{Bmatrix} \{X(0)\} \\ \{X(1)\} \\ \vdots \\ \{X(p-1)\} \end{Bmatrix}, \tag{5.91}$$

$$[h] = \begin{bmatrix} [h(0)] & [0] & \cdots & [0] \\ [h(1)] & [h(0)] & \cdots & [0] \\ \vdots & \vdots & \ddots & \vdots \\ [h(p-1)] & [h(p-2)] & \cdots & [h(0)] \end{bmatrix} \tag{5.92}$$

e

$$\{F\} = \begin{Bmatrix} F(0) \\ F(1) \\ \vdots \\ F(p-1) \end{Bmatrix} \tag{5.93}$$

Sendo que,

$$\{X(i)\} = \begin{Bmatrix} x_1(i) \\ x_2(i) \\ x_3(i) \end{Bmatrix}, \quad [h(i)] = \Delta t \begin{bmatrix} h_{11}(i) \\ h_{21}(i) \\ h_{31}(i) \end{bmatrix} \quad e \quad [0] = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.94}$$

com $i = 0, 1, \cdots, p-1$.

Para calcular o valor do vetor $\{F\}$ a partir da Eq. (5.90) será utilizado o método dos mínimos quadrados. Assim, tem-se que:

$$\{F\} = \left([h]^T [h]\right)^{-1} [h]^T \{X\} \tag{5.95}$$

Conforme apresentado por PURCINA (2010), os sinais de força e de acelerações, sinais de entrada e saída respectivamente, foram amplificados usando amplificadores de sinais e enviados para a placa de aquisição e posteriormente armazenados em um microcomputador. Cada sinal recebido foi observado no intervalo de 0 a 13,67s discretizado em 3500 pontos igualmente espaçados ($\Delta t = 3,9 \times 10^{-3}$s). A solução do sistema dado pela Eq. (5.90) será obtida para as F.R.I.s $h_{11}(t)$, $h_{21}(t)$ e $h_{31}(t)$ por meio da minimização do resíduo $r = |\{X\} - [h]\{F\}|$ pelo método EDMP.

Visto que a força exata é conhecida, esta será utilizada para comparar com

a força identificada a fim de avaliar a precisão do procedimento. Em seguida, serão apresentados nas Figs. 5.15 à 5.22 uma comparação entre os resultados encontrados e os valores exatos da força de excitação e da aceleração para cada uma das placas considerando os erros cometidos.

Na simulação realizada adotou-se 160 indivíduos na população inicial divididos entre 16 processadores do cluster assim, cada processador executou o algoritmo com 10 indivíduos. Além disso, foi adotado os seguintes parâmetros para EDMP: fator de diferença F = 0,8 e probabilidade de cruzamento CR = 0,5. O critério de parada adotado foi o número máximo de gerações da população e a verificação de sua estagnação.

O tempo de execução do EDMP foi 3,86 horas para um total de 22.000 avaliações da função objetivo. O valor do resíduo no ponto ótimo é $r = 0,0594$.

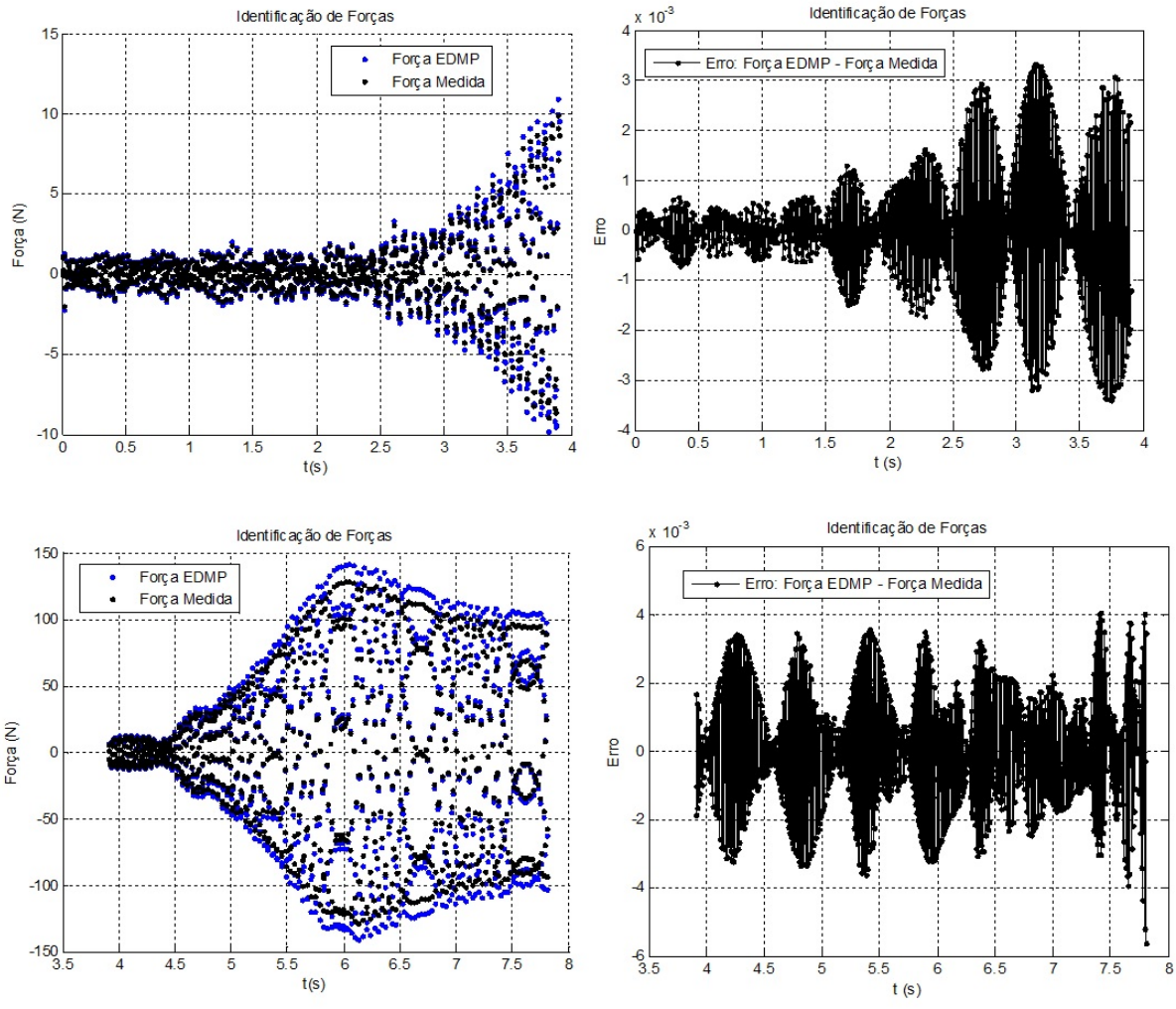


Figura 5.15 – Comparativo entre a Força aproximada por EDMP e os valores medidos para t=0s a t=8s.

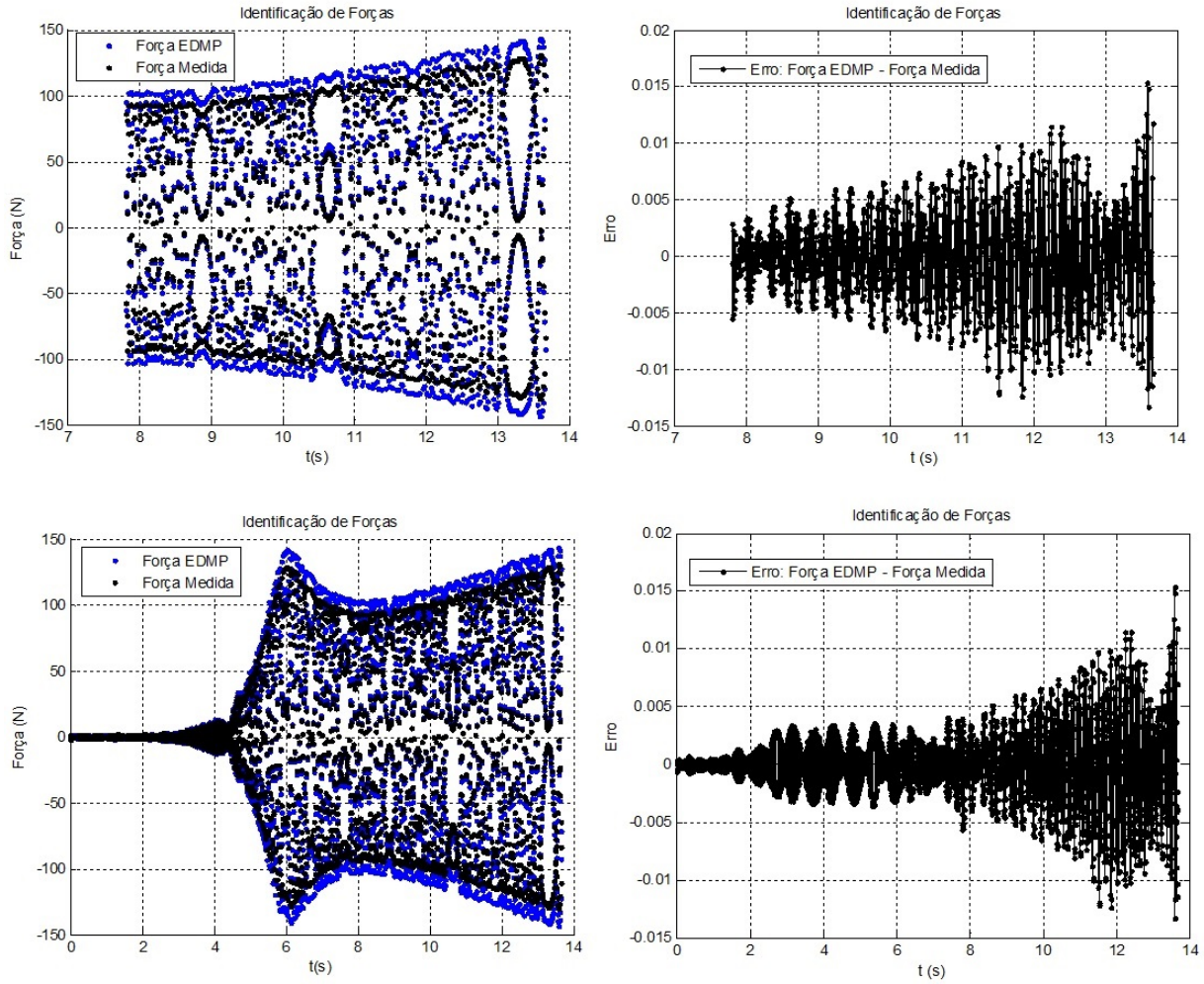


Figura 5.16 – Comparativo entre a Força aproximada por EDMP e os valores medidos para t=0s a t=14s.

Os resultados obtidos com EDMP foram muito satisfatórios já que o erro cometido é pequeno e aceitável. A maior vantagem de se usar EDMP é o fato de não se preocupar com condicionamento de matrizes e o da matriz não precisar ser quadrada. Além disso, o problema de identificação de forças dinâmicas foi resolvido sem usar nenhuma técnica de regularização da matriz de coeficientes, visto que esta é uma matriz retangular formada por três blocos triangulares.

Uma desvantagem do EDMP é o tempo de execução já que a solução de grandes sistemas lineares requer grandes recursos computacionais uma vez que o EDMP trabalha com avaliação da função objetivo. Além disso, visto que há um grande volume de dados para armazenar, a solução destes problemas precisa de um cluster com memória distribuída ou supercomputadores com grande memória compartilhada.

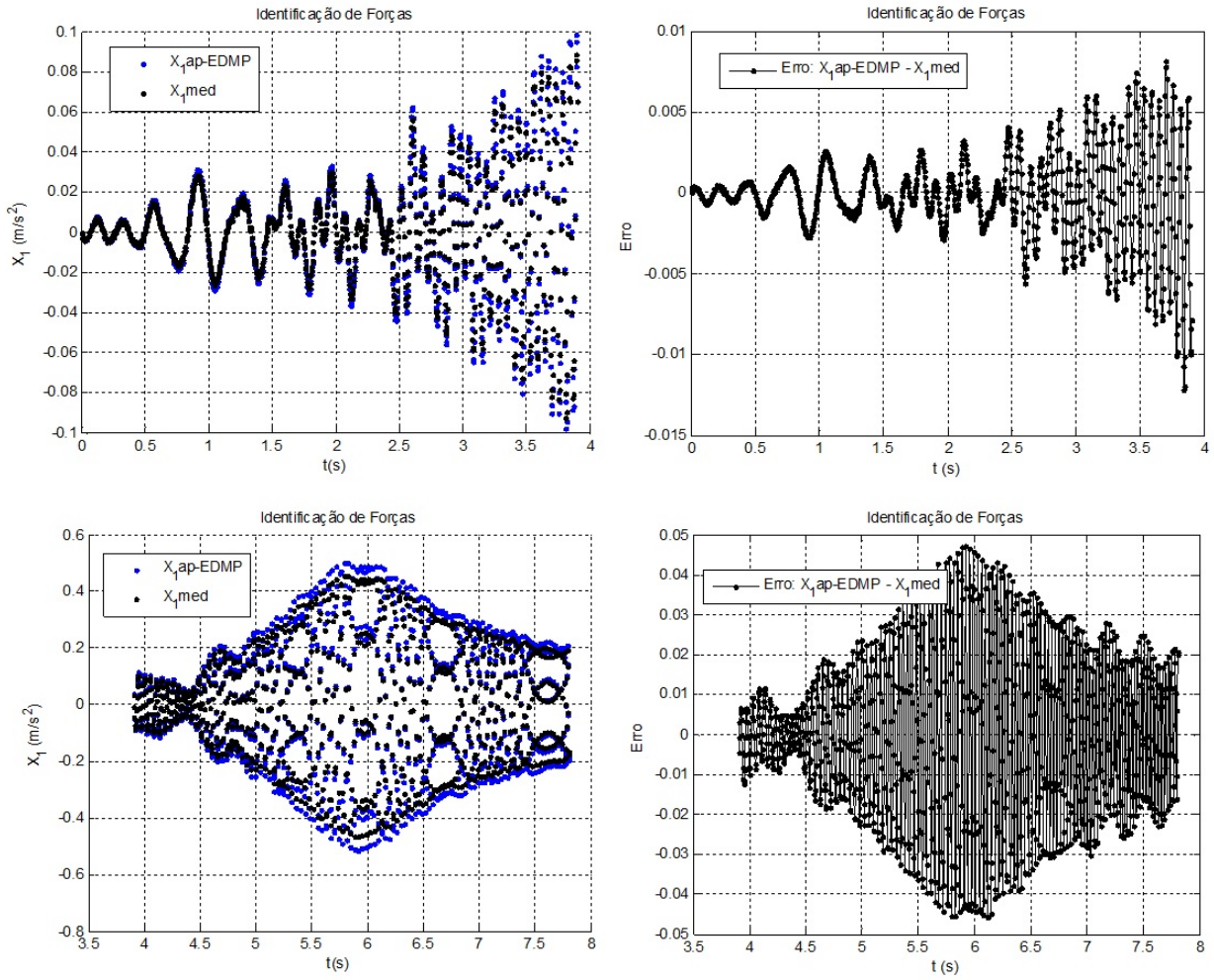


Figura 5.17 – Comparativo entre a resposta X_1 aproximada por EDMP e os valores medidos para t=0s a t=8s.

Visto que a solução de sistemas lineares desempenha um papel importante nas engenharias há muitos métodos numéricos para resolução sistemas lineares, mas muitas também são as suas limitações. Em RUGGIERO e LOPES (1996) pode-se encontrar um estudo das seguintes metodologias de solução de sistemas lineares: Fatoração LU, Fatoração Cholesky, Jacobi, Gauss-Seidel. Estes métodos são mais eficazes para matrizes quadradas. No trabalho de STRANG (2005) é feita uma análise dos métodos Gradiente Conjugado, GMRES e QR para Autovalores. Tais métodos, geralmente, são mais eficientes para matrizes simétricas positiva definida.

Embora a solução de sistemas lineares seja conceitualmente simples, na prática podem surgir muitos desafios. Acredita-se, devido as simulações realizadas nesta pesquisa, que o EDMP seja uma alternativa promissora, principalmente devido a pratici-

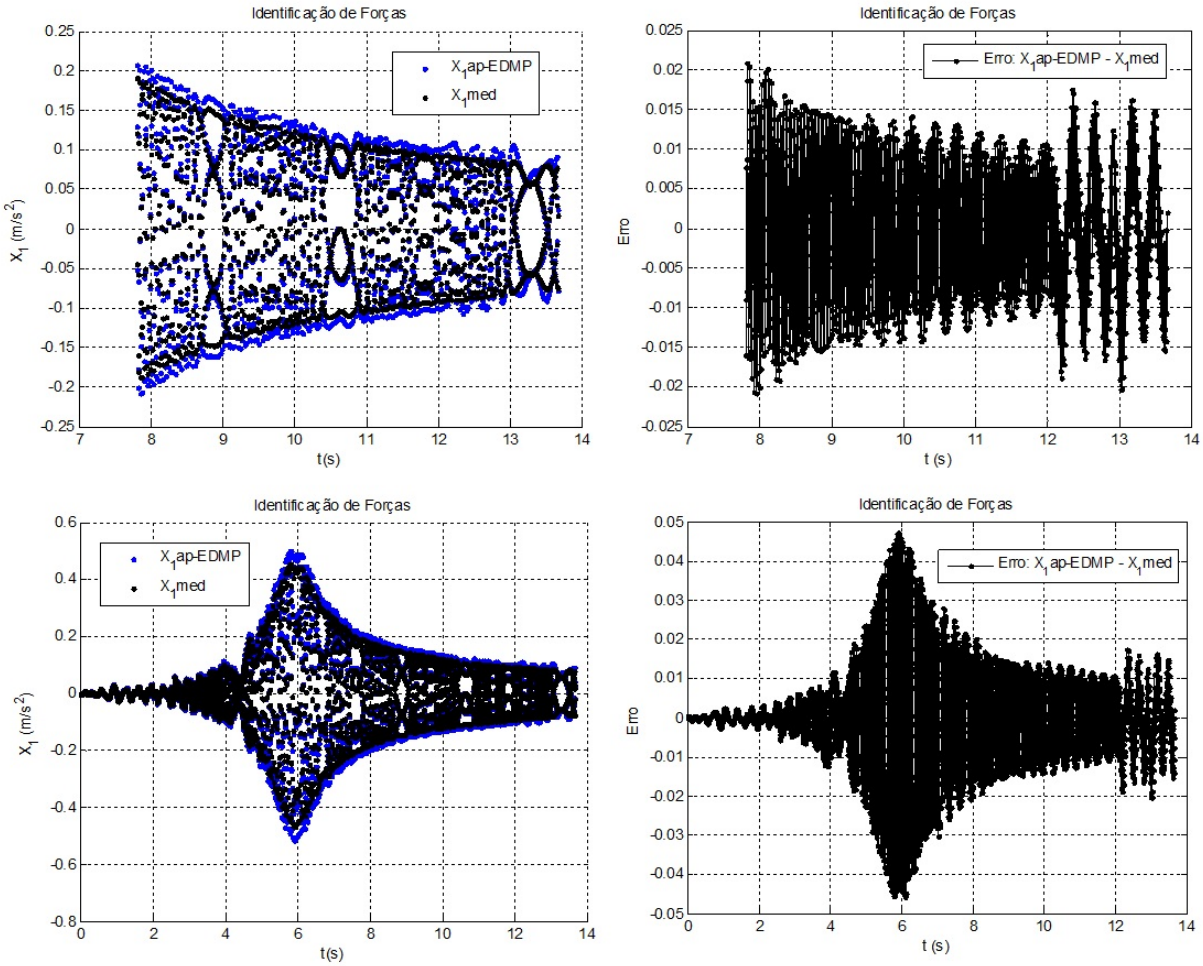


Figura 5.18 – Comparativo entre a resposta X_1 aproximada por EDMP e os valores medidos para t=0s a t=14s.

dade de não se preocupar com as características da matriz dos coeficientes do sistema linear.

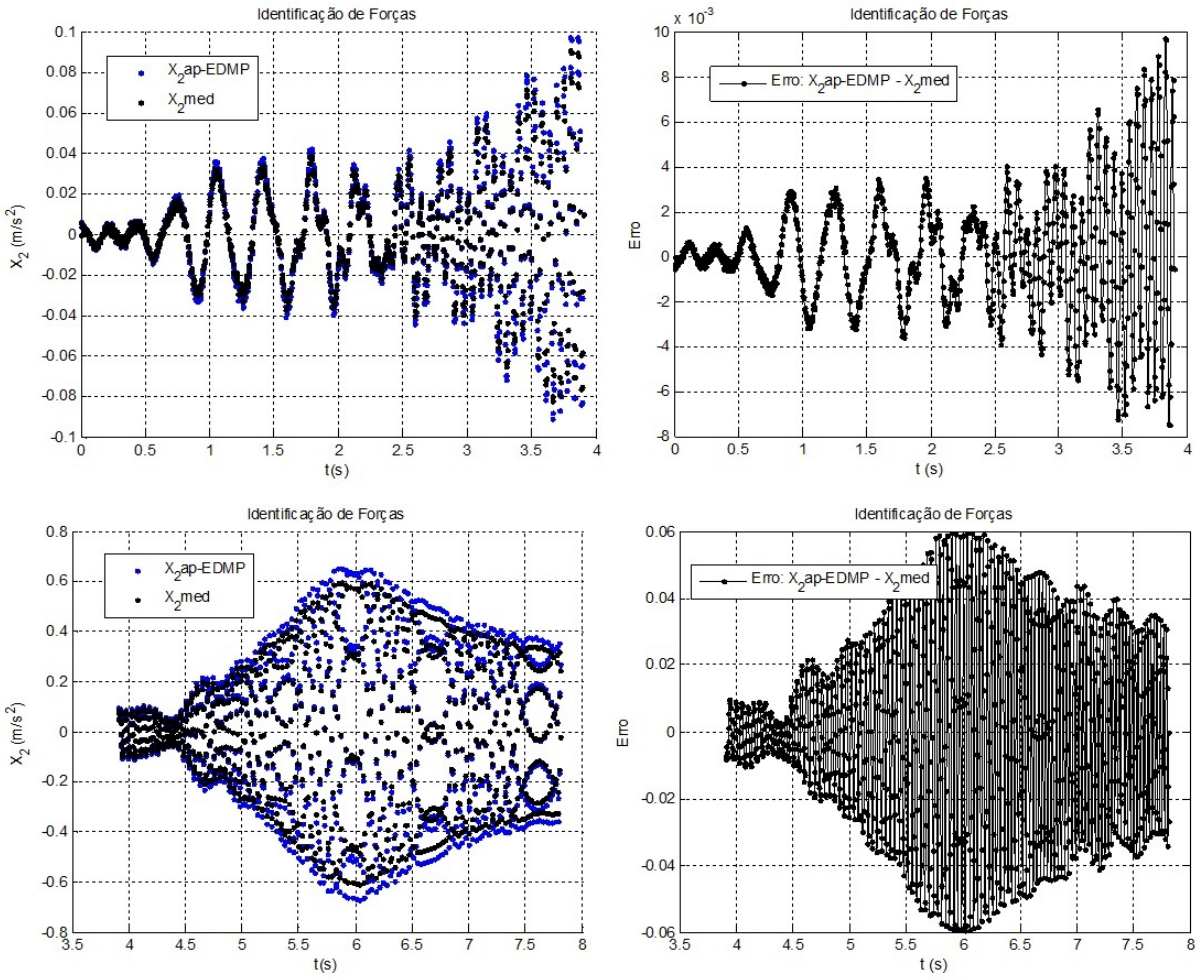


Figura 5.19 – Comparativo entre a resposta X_2 aproximada por EDMP e os valores medidos para t=0s a t=8s.

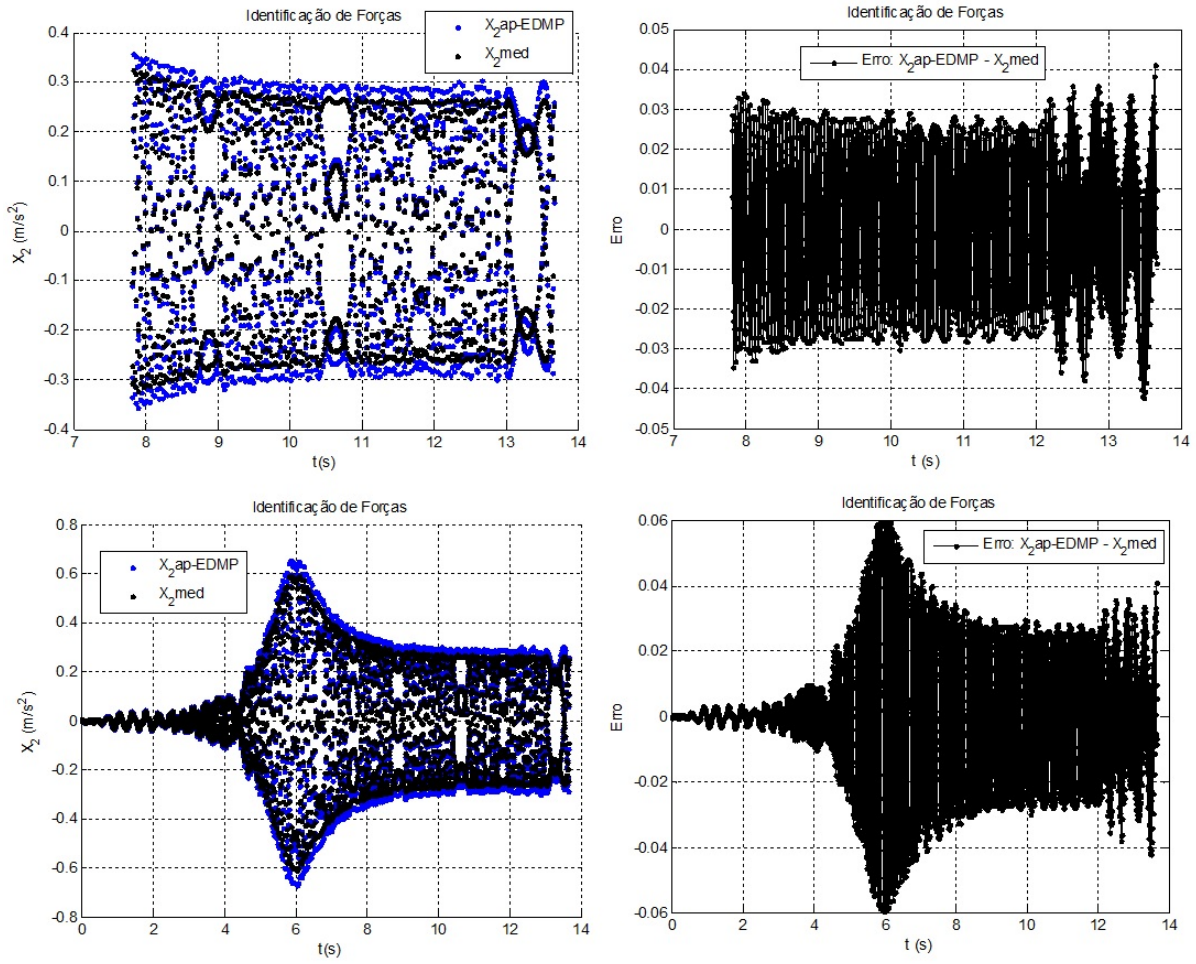


Figura 5.20 – Comparativo entre a resposta X_2 aproximada por EDMP e os valores medidos para t=0s a t=14s.

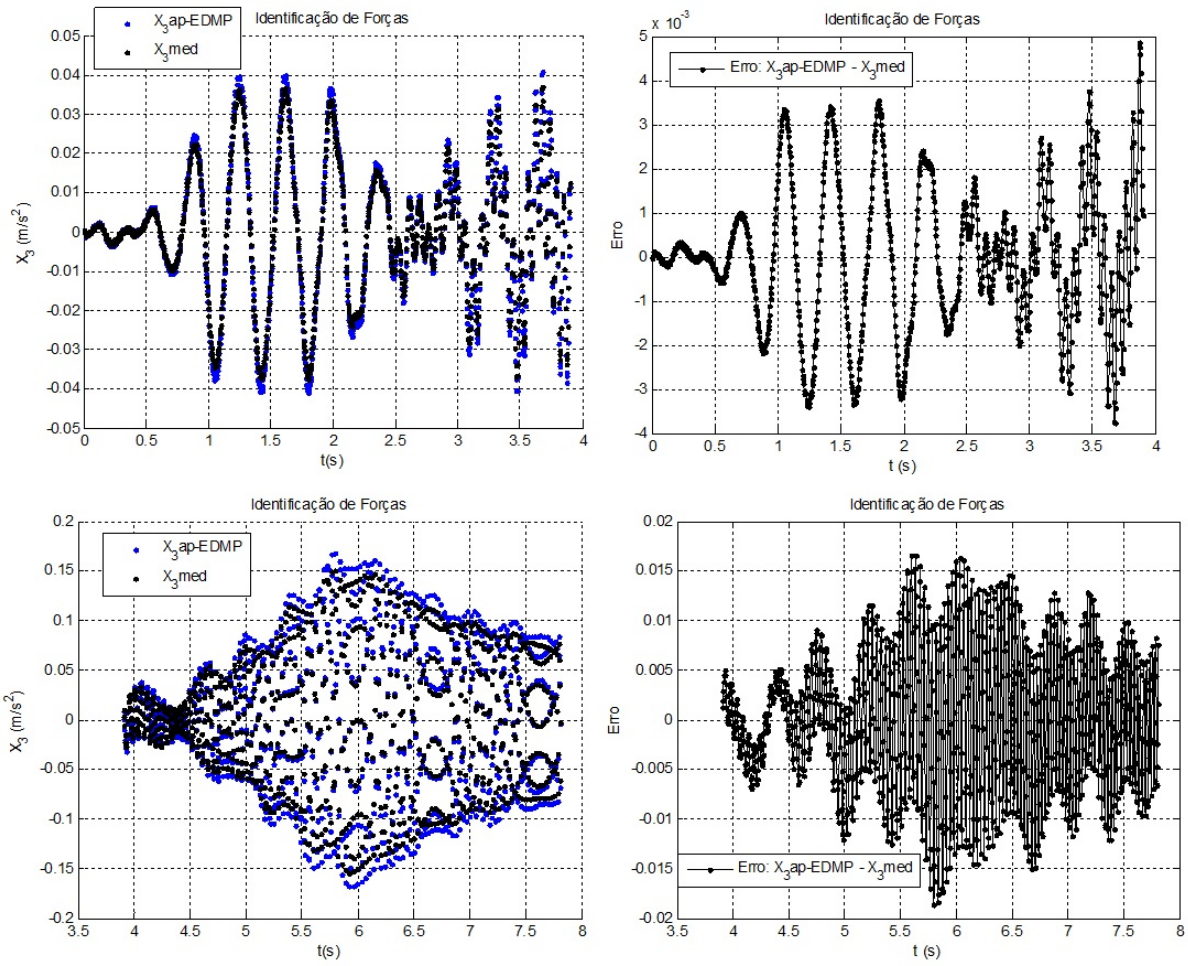


Figura 5.21 – Comparativo entre a resposta X_3 aproximada por EDMP e os valores medidos para t=0s a t=8s.

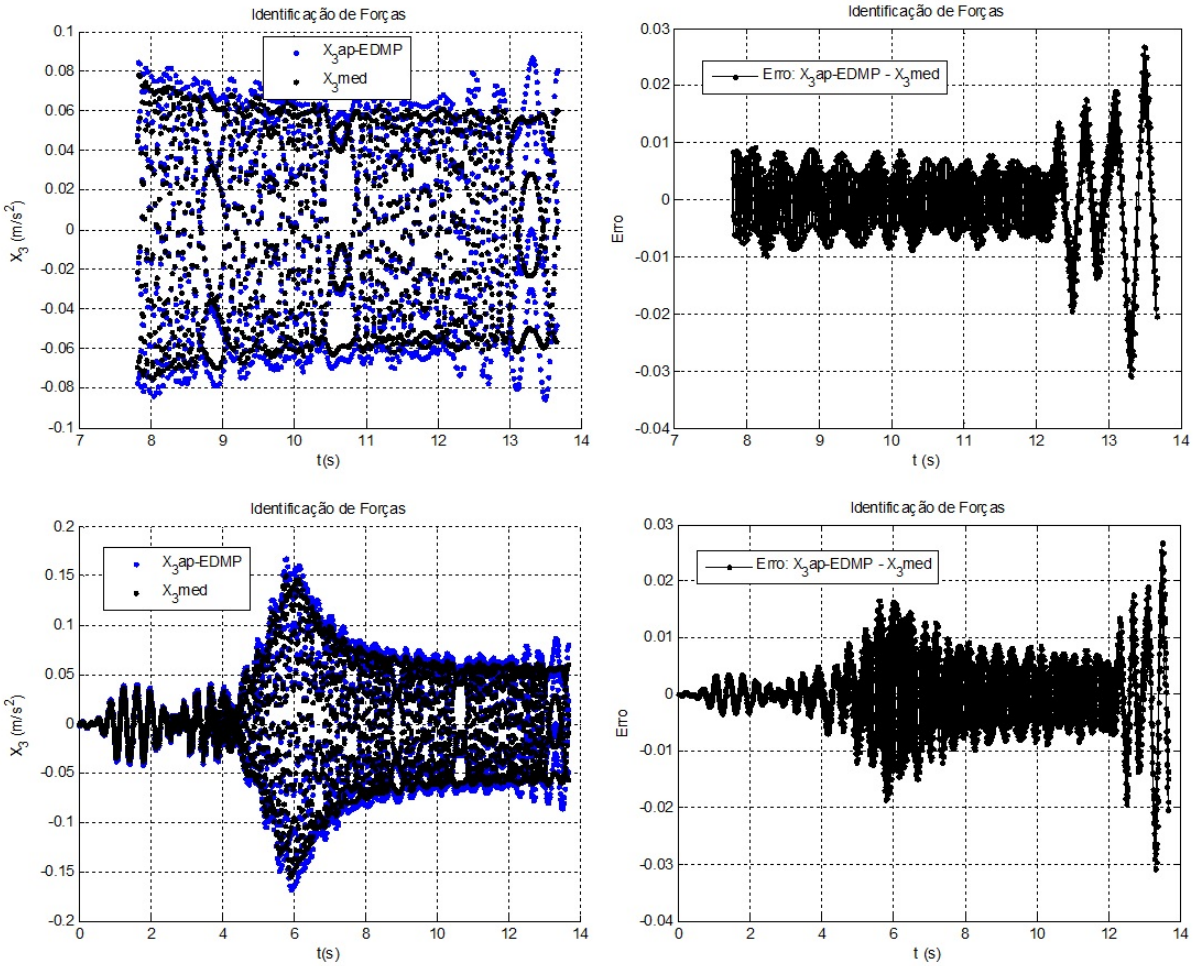


Figura 5.22 – Comparativo entre a resposta X_3 aproximada por EDMP e os valores medidos para t=0s a t=14s.

CAPÍTULO VI

CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, foi apresentado um aprimoramento do método de otimização evolutivo Evolução Diferencial por meio de modificações no seu algoritmo básico utilizando técnicas de conjuntos embaralhados (shuffled complex). O novo algoritmo foi implementado em computação paralela e recebeu o nome de Evolução Diferencial Melhorada (EDMP).

A EDMP introduz inovações quanto ao modo de evoluir a população inicial. No algoritmo original a população inicial sofre a evolução num único processador, no algoritmo modificado a população inicial é dividida em subpopulações que então evoluem separadamente em processadores diferentes. Esta modificação aumenta a diversidade da população, evita convergência prematura e acelera o tempo de execução.

A fim de validar o algoritmo várias simulações foram feitas. As primeiras, mais simples, ajudaram a detectar e corrigir erros de implementações e possibilitaram fazer alguns aprimoramentos no código computacional. Em seguida, foram realizadas simulações de otimização de sistema robóticos sem e com restrições e, por último, simulações para resolução de grandes sistemas lineares.

As análises das simulações realizadas comprovam que o algoritmo EDMP é eficiente e robusto. Além disso, confirmaram a intuição inicial de que grandes populações podem ser divididas em populações menores, com a busca realizada por meio

de processamento em paralelo. Devido aos avanços de *hardware* e *software* surgiram máquinas que executam tarefas com vários processadores permitindo a troca de informações com rapidez, o que resulta em economia de tempo e esforço computacional. Isto ficou evidente com EDMP, o tempo de processamento durante a execução do algoritmo melhorado é consideravelmente menor quando comparado à outros algoritmos tais como Algoritmos Genéticos, Evolução com Conjuntos Embaralhados e Evolução Diferencial.

Além disso, o algoritmo EDMP é relativamente simples e mostrou-se eficiente em diversos contextos tais como:

- otimização uni-objetivo restrita, como nos casos de obtenção dos projetos ótimos de um recipiente de pressão e de uma viga engastada;
- otimização uni-objetivo irrestrita, nos quais foram simulados as funções teste de Beale, Michalewicz, Levy, Shubert e Rastrigin, sistemas robóticos e solução de grandes sistemas lineares por meio do problema de identificação de forças dinâmicas e de um sistema dinâmico;
- otimização multiobjetivo restrita, no caso da otimização de sistemas robóticos considerando sua topologia,
- otimização multimodal, nos casos das funções testes de Shubert e Rastrigin, de sistemas robóticos considerando sua topologia.

Assim, entende-se que os objetivos desta tese foram cumpridos.

Portanto, a principal contribuição deste trabalho é apresentar uma ferramenta computacional chamada de Evolução Diferencial Melhorada implementada em paralelo capaz de resolver problemas de otimização por meio de computação paralela.

Uma outra contribuição importante é possibilitar ao projetista de sistemas robóticos a escolha da topologia do espaço de trabalho desejada e ajustar os parâmetros de Denavit-Hartenberg que satisfazem aos critérios propostos, ou seja, otimizando simultaneamente o volume, a rigidez e a destreza de manipuladores seriais 3R. Este trabalho foi uma extensão da pesquisa desenvolvida por OLIVEIRA (2011), com a finalidade de criar uma nova ferramenta capaz de resolver o problema proposto em um tempo com-

putacional viável para aplicações industriais. Nesta linha de pesquisa, os seguintes de estudos futuros, podem ser considerados:

- desenvolvimento e utilização de outros índices de desempenho, como por exemplo, performance dinâmica;
- estudo da robustez dos manipuladores 3R com eixos não ortogonais;
- extensão deste estudo aos manipuladores paralelos;
- estudo de outros métodos para lidar com problemas restritos considerando, por exemplo, a função de penalidade interior;
- uso de novas técnicas baseadas em dominância de Pareto para problemas multiobjetivos, incorporando, por exemplo, o conceito proposto por BANDYOPADHYAY et al (2008) de Arquivo ou de conjuntos de potenciais soluções ótimas de Pareto que armazenam as soluções não dominadas a fim de prover um conjunto de soluções para o problema.

Embora o algoritmo EDMP esteja pronto, acredita-se que este ainda possa ser aperfeiçoado a fim de aumentar sua eficiência quanto à solução de sistemas lineares de grande porte. Ainda que os resultados encontrados na solução de sistemas lineares sejam satisfatórios, o tempo de execução do algoritmo é alto. Considerando que a solução de grandes sistemas lineares é de grande importância para várias aplicações em engenharia, esta área de pesquisa tem muito ainda a ser explorada. Por isso, como proposta de continuidade deste trabalho, sugere-se:

- o desenvolvimento de um modelo de otimização híbrido envolvendo Evolução Diferencial Melhorada e uma técnica de busca local para aumentar a velocidade de convergência na solução de sistemas lineares de grande porte;
- a procura de novas formas para escrever o funcional a ser otimizado, bem como definir novos critérios de parada do algoritmo, visando melhorar a eficiência do mesmo.

No entanto, os resultados alcançados no presente trabalho apontam para ganhos significativos, o que justifica a continuidade da pesquisa.

Referências Bibliográficas

Agência Brasileira de Notícias, *SUSE Linux é usado por mais de 1/3 dos 100 melhores supercomputadores do mundo.* São Paulo, 2013. Disponível na Internet via <http://www.abn.com.br/editorias1.php?id=73841>. Arquivo capturado em 03 de setembro de 2014.

ARORA, J.S., *Introduction to Optimum Design.* McGraw-Hill Book Company, Singapore, Roma, 1987.

BABU B. V., MUNAWAR S. A., *Differential evolution strategies for optimal design of shell-and-tube heat exchangers.* Chemical Engineering Science, v. 62, p. 3720 - 3739, 2007.

BÄCK, T., HOFFMEISTER, F. SCHWEFEL, H.-P, *Survey of Evolution Strategies.* In L. B. Belew and Booker, R. K. (eds.), *Proc. of the 4th Int. Conf. on GAs*,pp. 2-9, San Diego, CA, Julho, 1991.

BAILI, M., *Analise et Classification de Manipulateur 3R à axes Orthogonaux,* Thèse de Doctorat - University of Nantes, France, 2004.

BANDYOPADHYAY,S., SAHA, S., MAULIK, U., DEB, K., *A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA,* IEEE trans. on Evolutionary Computation, 2008.

BEKEY, G.A., M.H. GRAN, A.E. SABROFF, A. WONG, *Parameter optimization by random search using hybrid computer techniques,* AFIPS Conf. Proc. 29, 191-200, 1966.

BERGAMASCHI, P. R., SARAMAGO, S. F. P., COELHO, L. S., *Comparative study of*

SQP and metaheuristics for robotic manipulator design. Applied Mathematics and Computation, v. 58, p. 1396-1412, 2008.

BERGER, M.J. and COLELLA, P., *Local adaptive mesh refinement for shock hydrodynamics.* J. Comput. Phys., v. 82, p. 64-84, 1989.

BORN, J., *Evolutionsstrategie zur Numerischen Lösung von Adaptations Aufgaben.* Dissertação, Humboldt-Universität, Berlim 1978.

BRAGA, C. G., *O uso de Algoritmos Genéticos para aplicação em Problemas de Otimização de Sistemas Mecânicos.* Dissertação de mestrado, Universidade Federal de Uberlândia, 1998.

BRANDÃO, M. A. L., SARAMAGO, S. de F. P., *Métodos Estocásticos de Otimização: Algoritmos Genéticos e Evolução Diferencial.* São Carlos, SP, SBMAC: Notas em Matemática Aplicada, v. 55, 2011.

BRANDÃO, M. A. L., DORICIO, J. L., LOBATO F. S., SARAMAGO, S. F. P., *A Comparative Study Using Shuffled Complex Evolution and the Differential Evolution Applied to Robotic Manipulator Design* . 10^{th} World Congress on Computational Mechanics, Computational Mechanics 2012 Proceedings, 2012.

BRANDÃO, M. A. L., SARAMAGO, S. F. P., DORICIO, J. L., *Optimum desing of 3R robot manipulator by using improved differential evolution implemented in parallel computation.* 22nd International Congress of Mechanical Engineering - COBEM, 2013.

BUENO, A. D., *Introdução ao Processamento Paralelo e ao Uso de Clusters de Workstacions em Sistemas GNU/LINUX Parte I: Filosofia.* Laboratório de Meios Porosos e Propriedades Termofísicas - LMPT 12 de novembro de 2002.

BUJOK PETR, *Parallel Models of Adaptive Differential Evolution Based on Migration Process.* Aplimat - Journal of Applied Mathematics, v. 4, n. 2, 2011.

CHI ZHOU, *Fast parallelization of differential evolution algorithm using MapReduce.* Proceedings of the 12th annual conference on Genetic and evolutionary computation, Portland, Oregon, USA, July 07-11, 2010. [doi>10.1145/1830483.1830689].

CHONG, C K., M S. MOHAMAD, S. DERIS, M S. SHAMSIR, Y W. CHOON, and L E. CHAI, *Improved Differential Evolution Algorithm for Parameter Estimation to Improve the Production of Biochemical Pathway.* International Journal of Interactive Multimedia and Artificial Intelligence. Special Issue on Distributed Computing and Artificial Intelligence, v.1, p. 22-29, 2012.

COELLO C. A. C., *Use of a self-adaptive penalty approach for engineering optimization problems.* Computers in Industry, v. 41, n. 2, p. 113-127, 2000.

COELHO, L. dos S., SOUZA, R. C. T., MARIANI, V. C., *Improved differential evolution approach based on cultural algorithm and diversity measure applied to solve economic load dispatch problems.* Mathematics and Computers in Simulation, v. 79, Issue 10, p. 3136-3147, June 2009.

COMPANY, O., PIERROT, F., FAUROUX, J. C., *A method for modeling analytical stiffness of a lower mobility parallel manipulator.* Proc. Of IEEE ICRA: Int. Conf. On Robotics and Automation, Barcelona, Spain, April 18-25, 2005.

DARWIN, C., *Die Abstammung des Menschen.* Translation of the 2nd rev. ed. of The descent of man, Kröner, Stuttgart, 1966.

DAS, S., KONAR, A. and CHAKRBORTY, U.K., *Two improved differential evolution schemes for faster global search.* In GECCO'05. Washington, DC, USA, 2005.

DAS S., SUGANTHAN P. N., *Differential Evolution: A Survey of the State-of-the-Art.* IEEE Trans. on Evolutionary Computation, 2011.

DEAN, J., GHEMAWAT, S., *MapReduce: Simplified Data Processing on Large Clusters.* Disponível na Internet via <http://research.google.com/archive/mapreduce.html>. Arquivo capturado em 23 de junho de 2014.

DEB, K., DASGUPTA, D. and MICHALEWICZ, *GenaAS: a robust optimal design technique for mechanical component design.* Evolutionary Algorithms in Engineering Applications Berlin, p. 497-514, 1997.

DEBLAISE, D., HERNOT, X., MAURINE, P., *A Systematic Analytical Method for PKM Stiffness Matrix Calculation.* IEEE International Conference on Robotics and Automation, 2006.

DEGRAAG, D.P., *Parameter optimization techniques for hybrid computers.* Proceedings of the VIth International Analogue Computation Meeting, Munich, Aug-Sept., p. 136-139, 1970.

DEKKER, T. J., HOFFMANN, W., POTMA, K., *Parallel algorithms for solving large linear systems,* Journal of Computational and Applied Mathematics 50, p. 221-232, 1994.

DUAN, Q., *Shuffled Complex Evolution (SCE-UA) Method.* Disponível na Internet via <http://www.mathworks.com/matlabcentral/fileexchange/7671-shuffled-complex-evolution–sce-ua–method>.

DUAN, Q. A., GUPTA, V. K. and SOROOSHIAN, S., *Shuffled Complex Evolution Approach for Effective and Efficient Global Minimization.* Journal of Optimization Theory and Applications: v. 76, n. 3, March 1993.

EL-KHASAWNEH, B. S., FERREIRA, P. M., *Computation of stiffness and stiffness bounds for parallel link manipulator.* J. Machine Tools & manufacture, v. 39(2), p. 321-342, 1999.

GOLDBERG, David E., *Genetic algorithms in search, optimization, and machine learning.* The University of Alabama, Addison-Wesley Publishing Company, INC, 1989.

GONZALEZ, R.S., *An optimization study on a hybrid computer.* Ann. Assoc. Int'l Calcul Analog. 12, p. 138-148, 1970.

HEITKOETTER J. and BEASLEY D., *The Hitch-Hiker's Guide to Evolutionary Computation: A list of Frequently Asked Questions.* USENET:comp.ai.genetic, 1994.

HOLLAND, J. H., *Adaptation in Natural and Artificial Systems: An introductory analysis with applications to biology, control, and artificial intelligence.* Oxford, England: University of Michigan Press, Ann Arbor, Michigan, viii, p. 183, 1975.

HOUCK, C.R., JOINEZ, J.A. and KAY, M.G., *A Genetic Algorithms for Function Optimization: a Matlab Implementation.* NCSO-IE Technical Report, University of North Caroline, USA, 1985.

HSIEH, J., *High-Performance Computing with Beowulf Clusters.* DELL Power Solutions, p. 75-79, June, 2000. Disponível na Internet via <www.dell.com/powersolutions.>. Arquivo capturado em 27 de agosto de 2011.

HU, X.; EBERHART, R.; SHI, Y., *Engineering optimization with particle swarm.* IEEE Conference on Swarm Intelligence, Indianapolis, Indiana, 2003.

HWANG K., *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill, ISBN 0-07-031622-8, 1993.

KWEDLO, W., BANDURSKI, K., *A Parallel Differential Evolution Algorithm for Neural Network Training.* Parallel Computing in Electrical Engineering. PAR ELEC 2006.

LEON S. J., *Algebra Linear: com aplicações.* 8^aed, Rio de Janeiro: LTC, 2011.

LI, R., XU, L., SHI, X.-W., ZHANH, N., and LV, Z.-Q., *Improved differential evolution strategy for antenna array pattern synthesis problems.* Progress In Electromagnetics Research, v. 113, p. 429-441, 2011.

LINS R. C., XAVIER R. S., MADEIRO F., FERREIRA T. A. E., *Estimação de Movimento com Uso de Algoritmo Genétic Modificado.* VIII Simpósio Brasileiro de Automação Inteligente, Florianópolis, Santa Catarina. Disponível na Internet via <http://www.sba.org.br/rsv/SBAI/SBAI2007/docs/60100068.pdf>. Arquivo capturado em 23 de junho de 2014.

LOBATO F. S., OLIVEIRA-LOPES L. C., MURATA V. V., STEFFEN JR. V., *Solution of Multiobjective Optimal Control Problems with Index Fluctuation using Differential Evolution.* 6th Brazilian Conference on Dynamics, Control and Applications - DINCON, 2007.

LOBATO F. S., STEFFEN JR. V., *Engineering System Design with Multi-objective Differential Evolution.* 19th International Congress of Mechanical Engineering - COBEM 2007.

MARIANI V. C., LIMA A. G. B., COELHO L. S., *Apparent Thermal Diffusivity Estimation of the Banana During Drying using Inverse Method.* Journal of Food Engineering, v. 85, p. 569-579, 2008.

MENDES S. P., PULIDO J. A. G., RODRIGUEZ M. A. V., SIMON M. D. J., PEREZ J. M. S., *A Differential Evolution Based Algorithm to Optimize the Radio Network Design Problem.* Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, p.119, December 04-06, 2006.

MORINOTO, C. E., *Dicionário Técnico de Informática 3ed.* Disponível em <http://www.guiadohardware.net>.

MUSRRAT, Ali, PANT, M., ABRAHAM, A., *Simplex Differential Evolution.* Acta Polytechnica Hungarica, v.6, n.5, 2009.

NELDER, J. A., and MEAD, R., *A Simplex Method for Function Minimization.* Computer Journal, v. 7, p. 308-313, 1965.

NETO P. S. G. M., PETRY G. G., ATAIDE J. P. M., FERREIRA T. A. E., *Combinação de Redes Neurais Artificiais com Algoritmo Genético Modificado para a Previsão de Séries Temporais.* XXV Congresso da Sociedade Brasileira de Computação, Unisinos, São Leopoldo - RS, 2005.

NOCEDAL, J., WRIGHT, S.J., *Numerical Optimization.* Springer Series in Operations Research, 2000.

NTIPTENI, M. S., VALAKOS I. M., NIKOLOS I. K., *An asynchronous parallel differential evolution algorithm.* Proc. ERCOFTAC Conf. Design Optimization: Methods Applicat, 2006.

OCAMPO E. M. T., GRISALES Y. S. R., ECHEVERRI M. G., *Algoritmo genetico modificado aplicado al problema de secuenciamiento de tareas en sistemas de produccion lineal - flow shop.* Scientia Et Technica, v. XII, n. 30, p. 285-290, Universidad Tecnológica de Pereira Colombia, 2006. Disponível na Internet via

<http://www.redalyc.org/articulo.oa?id=84920491003>. Arquivo capturado em 23 de junho de 2014.

OLIVEIRA, L. S. de., *Uma contribuição ao estudo dos métodos de otimização multiobjetivo.* Dissertação (mestrado) - Universidade Federal de Uberlândia Programa de Pós-Graduação em Engenharia Mecânica, 2005.

OLIVEIRA, G. T. S., *Estudo e Aplicações da Evolução Diferencial.* Dissertação de Mestrado - Universidade Federal de Uberlândia, Uberlândia, MG, Brasil, 126p, 2006.

OLIVEIRA, G.T.S.,PRADO, J. R., SARAMAGO, S.F.P., BERGAMASCHI, P. R., *Maximização do Espaço de Trabalho de Manipuladores 3R usando Algoritmos Evolutivos.* In: XXVII Iberian Latin-American Congress on Computational Methods in Engineering (CILAMCE), Belém do Pará, Brasil. Anais do XXVII CILAMCE (CD-ROM). Associação Brasileira de Mecânica Computacional - ABMEC, 2006.

OLIVEIRA, G.T.S., NETO, A.D.C., MENDONÇA, S.A., SARAMAGO, S.F.P., *Projeto Ótimo de Manipuladores 3R Ortogonais considerando a sua Topologia.* Proceedings of the 5th Congresso nacional de engenharia Mecânica - CONEM, Associação Brasileira de Mecânica Computacional - ABMEC, Salvador, Brasil, 2008.

OLIVEIRA, L. S., SARAMAGO, S. F. P., *Multiobjective Optimization Techniques Applied to Engineering Problems.* Journal of the Brazilian Society of Mechanical Sciences and Engineering (Impresso). v.XXXII, p.94 - 104, 2010.

OLIVEIRA, G.T.S., *Projeto ótimo de robôs manipuladores 3r considerando a topologia do espaço de trabalho.* Tese (doutorado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Mecânica, 2011.

OPTIMA *Global optimization test problems.* Disponível na Internet via <http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm>. Arquivo capturado em 21 de agosto de 2014.

PAUL, R. P., STEVENSON, C. N., *Kinematics of robot wrists.* The Int. J. Robotics Res. 2, n. 1, p. 31-38, 1983.

PLN, *Grupo de Pesquisa em Processamento da Linguagem Natural da Pontifícia Universidade Católica do Rio Grande do Sul - Porto Alegre - RS - Brasil,* Disponível na Internet via <http://www.inf.pucrs.br/∼linatural/index.html>. Arquivo capturado em 07 de novembro de 2014.

PRICE, W. L., *A Controlled Random Search Procedure for Global Optimization,* Toward Global Optimization 2, Edited by L. C. W. Dixon and G. P. Szeg6, North-Holland, Amsterdam, Holland, p. 71-84, 1978.

PRICE, W. L., *Global Optimization by Controlled Random Search.* Journal of Optimization Theory and Applications, v. 40, p. 333-348, 1983.

PRICE, W. L., *Global Optimization Algorithms for a CAD Workstation.* Journal of Optimization Theory and Applications, v. 55, p. 133-146, 1987.

PURCINA, L.A. and SARAMAGO, S.F.P., *Differential evolution applied to the solution of large linear systems.* In EngOpt 2008 - International Conference on Engineering Optimization. Rio de Janeiro, Brazil, 2008.

PURCINA, L. A., *Técnicas de Otimização Evolutivas aplicadas à Solução de Grandes Sistemas Lineares.* Tese de doutorado. Universidade Federal de Uberlândia, 2010.

QIANG, J., CHAO Y., MITCHELL C. E., PARET S., RYNE R. D., *A Parallel Multi-Objective Differential Evolution Algorithm for Photoinjector Beam Dynamics Optimization.* Proceedings of IPAC2013, Shanghai, China, 2013.

QUARANTA G., FIORE A., MARANO G. C., *Optimum design of prestressed concrete beams using constrained differential evolution algorithm.* Structural and Multidisciplinary Optimization, v. 49, Issue 3, p. 441-453, 2014.

QUENTAL, N. C., *Modelagem de Desempenho de Programas Paralelos Utilizando Redes de Petri Temporizadas.* Trabalho de Conclusão de Curso em Engenharia da Computação pela Escola Politécnica de Pernambuco - Universidade de Pernambuco, 2006.

RAUPP, F. M. P., FAMPA, M. H. C., MELO, W. A. X., *Evolução Diferencial Aperfeiçoada para otimização contínua restrita.* XLII SBPO, Bento Gonçalves-RS, 2010.

RECHENBERG, I., *Evolutionsstrategie-Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart, 1973.

ROCHA R., *Notas de Aula.* Disponível na Internet via <http://www.dcc.fc.up.pt/ ricroc/aulas/0708/ppd>. Arquivo capturado em 24 de junho de 2014.

RUA, C.M., *Simulação computacional adaptativa de escoamentos bifásicos viscoelásticos.* Tese de Mestrado, IME - Universidade de São Paulo, 2013.

RUGGIERO, M. A., LOPES, V. L. R., *Cálculo Numérico - Aspectos Teóricos e Computacionais.* 2ª Edição. São Paulo: Makron Books, 1996.

SAAD, Y., SCHULTZ, M. H., *Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems,* SIAM Journal on Scientific and Statistical Computing 7(3), p. 856-869, 1986.

SANTOS R. R., SARAMAGO S. F. P., STEFFEN Jr, V., *Otimização do Torque Aplicado pelos Atuadores de Robôs Usando Técnicas de Controle Ótimo.* 15º POSMEC - Simpósio do Programa de Pós-Graduação em Engenharia Mecânica. Universidade Federal de Uberlândia, 2005.

SARAMAGO, S. F. P., BERGAMASCHI, P. R., OLIVEIRA, G. T. S., *Optimization of the Workspace Volume of 3R Manipulators Using Hybrid Methodologies.* 19th International Congress of Mechanical Engineering, Cobem, ABCM, v.1, p.1 - 10, 2007.

SCHWEFEL, H.-P., *Optimum Seeking Methods-User's Guides, Internal report of the Programme Group of Systems Analysis and Technological Development.* KFASTE-IB-7/81, Nuclear Research Center (KFA) Jülich, Germany, Oct. 1981.

SCHWEFEL, H.-P., *Collective Phenomena in Evolutionary Systems.* Preprints of the 31st Annual Meeting for General Systems Research, Budapeste, 2, pp. 1025-1033, 1987.

SCHWEFEL, H.-P., *Evolution and optimum seeking.* A Wiley-Interscience publication, New York, USA, 1995.

SOARES, G. L., *Algoritmos Genéticos: Estudo, Novas Técnicas e Aplicações.* Monografia de Curso de Especialização, Universidade Federal de Minas Gerais, Belo Horizonte, Brasil, 1997.

STALLINGS W., *Computer Organization and Architecture Designing for Performance.* Pearson Prentice Hall, ed.8 , 2010.

STEWART, E.C., W.P. KAVANAUGH, D.H. BROCKER, *Study of a global search algorithm for optimal control.* Proceedings of the Vth International Analogue Computation Meeting, Lausanne, p. 207-230, Aug.-Sept. 1967.

STORN R., *Differential Evolution Design of an IIR-filter with Requirements for Magnitude and Group Delay.* International Computer Science Institute, TR-95-026, 1995.

STORN, R., *On the Usage of Differential Evolution for Function Optimazation.* Biennal Conference of the North American Fuzzy Information Processing Society (NAFIPS,1996), Berkeley, p. 519-523, IEEE, 1996.

STORN, R. and PRICE, K., *Differential Evolution A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces.* Journal of Global Optimization, v. 11, p. 341-359, 1997.

STORN R. M., PRICE K. V., LAMPINEN J. A., *Differential Evolution - A Practical Approach to Global Optimization.* Springer, Natural Computing Series, 2005.

STRANG, G., *Mathematical Methods for Engineers II - Solving Large Linear Systems, 2005.* Disponível via Internet em <http://archive.org/details/flooved1698>. Arquivo capturado em 29 de agosto de 2014.

SURIBABU, C. R., *Differential evolution algorithm for optimal design of water distribution networks.* Journal of Hydroinformatics, v. 12.1, 2010.

SWAGATAM DAS, AMIT KONAR, UDAY K. CHAKRABORTY, *Two Improved Differential Evolution Schemes for Faster Global Search.* GECCO′05, June 25-29, Washington, DC, USA, 2005.

TASOULIS, D.K., PAVLIDIS, N.G., PLAGIANAKOS, V.P., VRAHATIS M.N., *Parallel differential evolution.* Congress on Evolutionary Computation (CEC 2004), p. 2023-2029, 2004.

TAKAHASHI, R. H. C., *Otimização Escalar e Vetorial.* Notas de Aula, v. 3, p. 223-236, Jan. 2007.

TEIXEIRA, O. N., LOBATO, W. A. L., YANAGUIBASHI, H. S., CAVALCANTE, R. V., SILVA, D. J. A., OLIVEIRA, R. C. L., *Algoritmo Genético com interação social na resolução de problemas de otimização global com restrições.* Computação Evolucionária em Problemas de Engenharia, ISBN 978-85-64619-00-5, 2011.

TOMASSINI, M., *Parallel and distributed evolutionary algorithms: A review.* Disponível na Internet via <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.924>. Arquivo capturado em 03 de setembro de 2014.

TOP500 Disponível na Internet via <http://www.top500.org/list/2014/06/>. Arquivo capturado em 24 de junho de 2014.

VANDERPLAATS, G., *Numerical Optimization Techniques for Engineering Design.* McGraw-Hill, USA, 1984.

VINOGRADOV, I. B., KOBRINSKI, A. E., STEPANENKO, Y. E., TIVES, L. T., *Details of kinematics of manipulators with the method of volumes (in Russian).* Mekhanika Mashin, n. 27-28, p.5-16, 1971.

WHITE, R.C., Jr., *Hybrid-computer optimization of systems with random parameters.* Ph.D. thesis, University of Arizona, Tucson AZ, June 1970.

APACHE^{TM} HADOOP®, Disponível na Internet via <http://hadoop.apache.org/>. Arquivo capturado em 07 de novembro de 2014.

YANG, D. C., LAI, Z. C., *On the conditioning of robotic manipulators-service angle.* ASME J. Mechanisms, Transm., and Auto. in Design 107, p. 262-270, 1985.

YOSHIKAWA, T., *Manipulability of robotic mechanisms.* The Int. J. Robotics Res. 4, n. 2, p.3-9, 1985.